2

FUNDING PROJECT NO. IT665702D625

TECOM PROJECT (TRMS) NO. 7-CO-RDO-EP1-004

TEST ACTIVITY REPORT NO. USAEPG-FR-1263

TEST SPONSOR: US ARMY TEST AND
EVALUATION COMMAND

METHODOLOGY INVESTIGATION

FINAL REPORT

PROGRAM FLOW ANALYZER

VOLUME II

BY
LESLIE F. CLAUDIO

AUGUST 1984

DISTRIBUTION UNLIMITED

US ARMY ELECTRONIC PROVING GROUND
Fort Huachuca, Arizona 85613-7110

APPENDIX E

PFA A-LEVEL SPECIFICATION

PROGRAM FLOW ANALYZER

A-LEVEL SPECIFICATION

(REVISED)

JANUARY 1984

USAEPG

TABLE OF CONTENTS

TABLES

1. SCOPE

This specification establishes the performance, design, development, and test requirements for the Program Flow Analyzer (PFA) system. PFA is a software analysis system that shall interpret and translate source language programs into attributes which characterize the design of the programs. The attributes shall then be processed to provide an analysis of the program based on selected PFA requirements. This specification provides the user requirements for a software analysis tool to support the software development process. Many of these requirements have not matured to the point of practical application. This document should serve as a living document to evolve as software engineering matures.

2. APPLICABLE DOCUMENTS

The following documents form a part of this specification to the extent herein specified. In the event of a conflict between the documents referenced herein and the contents of this specification, the contents of this specification shall be considered as superseding requirements.

2.1 Government Documents

Standards

Military MIL-STD-490	Military Standards Specification Practices, 30 October 1968
MIL-STD-1679	Military Standard Weapon System Software Development, 1 December 1978

Other Publications

USAEPG	Program Flow Analyzer Trade-Off Analysis

3. REQUIREMENTS

3.1 System Definition

The PFA is a software system which interprets and translates source language programs into data files which represent the design of the software program being analyzed.

3.1.1 PFA General Description

The PFA is a tool used for the analysis of a software program in order to identify software quality, program structure, and program maintainability. The design of the PFA shall include the following functional areas:

a. Program Structure Analysis

b. Data Flow Analysis

c. Finite State Analysis

d. Error Analysis

e. Program Logic Analysis

f. Software Quality Analysis

g. Software Change Analysis

h. Documentation Development and Review

3.1.2 System/Interface

The following paragraphs identify and describe the external and internal interfaces for the PFA system.

3.1.2.1 External Interface

File management and some segments of analysis and reporting use interactive commands. Files from previous runs shall be archived and used for comparison with files from new runs. The user shall supply, on a file, the source data to be analyzed. All functions shall generate reports to be printed.

3.1.2.2 Internal Interface

All functions of the system shall be independent and shall interface through the data files. The following files shall be used for functional interface.

3.1.2.2.1 Software Standards File

This file is created by the user and is used by analysis and reporting. The information contained in the software standards file shall specify standards for comparison to the software under test by software quality analysis.

3.1.2.2.2 Master File

The master file contains all information on the structure of the system being analyzed. It is created by the encoding function, maintained by file management, and used by the analysis and reporting function. The master file may be archived, and archived master files may serve as inputs to the analysis and reporting function.

3.1.2.2.3 User Files

User files (e.g., Selection File) are used for additional, user-defined requirements in analysis and reporting.

3.1.2.3 Host Operating System

The host computer shall be driven by the support software operating system peculiar to the machine being used for PFA. The operating system with all its utility routines (i.e., file manager, editor, math/statistical package, tape input/output (I/O) handlers, disk I/O handlers, and language compiler) shall be available in order that a request for services can be provided through the use of operating system directives. The host computer shall also be capable of backing up the PFA system files to tape.

3.1.3 Government Furnished Property (Target Hardware)

Initial implementation of the PFA shall be on the DEC-10 and VAX 11/780 systems at Fort Huachuca. Delivered software should be portable to other computers.

3.1.4 Operational and Organizational Concepts

The mission of PFA in relation to other systems under development is to assess the quality of the software system under development and to provide information for testing that software system.

Anticipated use of PFA will be at testing installations to include Post Deployment Software Support Centers (PDSSC's). PFA shall also be useable by developers.

3.2 Characteristics

3.2.1 Performance Characteristics

PFA shall read a representation of the system to be analyzed, maintain a master data base file containing parameters of that system, perform analysis, and generate reports on those parameters.

3.2.1.1 Program Structure Analysis

Program structure analysis should consist of several subfunctions:

a. Subroutine Call Structure Chart. A subroutine call structure chart shall show the subroutire call hierarchy, from the highest level routine to lowest level routine. External and recursive routines should also be identified.

b. HIPO Charts. A HIPO chart shall be generated for each module in the system being analyzed. The HIPO chart should contain the name of the module, a list of variables input to each program step, a list of subroutines called by each program step, a short description of each program step, and a list of variables output from each program step.

c. Execution Flow Diagram. An execution flow diagram shall provide a graphic representation of module control coupling and execution flow within each module of the software being analyzed. This graphic representation should be in the form of a bubble chart. These graphics should be machine generated and should show the control structure of the software unit under test.

d. Path Analysis. Path analysis shall look at all paths based on a node-branch diagram for the program segment and provide a list of all paths and path quantifiers, i.e., times, statement counts, and shortest or longest path. The path analysis shall also identify most common path lengths used, loop identification which describes the depth of nesting used, and path jumps into and out of loops. Path analysis shall also find the critical paths through the module.

3.2.1.2 Data Flow Analysis.

Data flow analysis should consist of several subfunctions which perform the following analysis and provide the appropriate reports:

a. Data Flow Diagram. A data flow diagram shall provide graphic representation of data flow between modules of the software being analyzed. This graphic representation should be in the form of a bubble chart with variables itemized on connecting lines showing directional data flow. These graphics should be machine generated using a plotter.

b. Module Coupling Analysis. Module coupling is a metric that shall provide information on the design of the target code. Module coupling is a metric for defining module interdependence for the sharing of control and data. Module coupling is based on intermodule control and data exchange. The control and data coupling between each module and every other module can be established and a numeric value can be assigned for each module-to-module relationship. The metric can then be used to report the coupling strength of the system under test.

c. Module Strength Analysis. Module strength (also known as cohesion) is another metric for defining module interdependence and functional design that shall provide information on the design of the target code. When the intramodule strength has been determined, a numeric value can be assigned.

d. Program Stability Analysis. Module coupling and module strength shall be used to calculate program stability, a software metric that describes the dependency of each module and its relationship with all other program modules. The program stability metric will report the probability for which a change in module X will have an effect in module Y.

e. Module-Level Data Flow Analysis. Module-level data flow analysis shall show the data flow between each step of execution in each module. This should be a graphic presentation similar to the data flow diagram in item a.

f. Critical Variable Analysis. A critical variable analysis shall be used to find which variables are used on the critical path of the module and ranks variables as to their criticality within the module.

g. Global Cross-Reference. A global cross-reference shall list all global variables used in the system, alphabetically and by the module in which they are used.

3.2.1.3 Finite State Analysis

Finite-state modeling should be used for locating deadlocked code, infinite loops, unexpected halts, and areas with oscillating states (cyclic actions or processing with transient intermediate states).

3.2.1.4 Error Analysis

Error analysis should determine (see paragraph 3.2.1.2) the effects of errors in data flowing between the modules. This should give a comprehensive picture of how and where the system being analyzed may fail.

3.2.1.5 Program Logic Analysis

Program logic analysis should consist of three subfunctions:

a. Simulation/Modeling Inputs Generation. The PFA shall be capable of generating a simplified module structure by combining units and paths within units. The structure shall be processable to produce input to a simulation language. The structure shall be capable of tracking input/output operations for each module and the variables which are affected by the decisions in the model design.

b. Interface to Commercially Available Tools. Commercially available tools should be evaluated to determine what analysis reports are available and what their inherent capabilities are in order to enhance the PFA system with their attributes or use their outputs to complement the PFA analysis reports. Interface programs should then be designed to extract and process information generated by the commercially available tools.

c. Symbolic Execution Analysis. Dynamic symbolic execution shall simulate the execution of each module and attempt to execute all possible paths through each module. This should provide an exhaustive test of each module.

3.2.1.6 Software Quality Analysis

The software quality analysis (SQA) function should consist of these subfunctions:

a. Software Profile. The software profile subfunction shall be capable of accepting user-defined software quality standards for the purpose of comparing the current program with these standards (e.g., MIL-STD-1679).

The software profile subfunction should include the following quantifiable data provided by the encoder function:

1. Number of documentation comments in the information prologue of module.

2. Number of documentation comments embedded in the body of the module.

3. Number of lines in the module containing more than one executable statement.

4. Number of executable statements per procedure.

5. Number of declaration (nonexecutable, non-comment) statements per module.

6. Number of control statements per module.

These metrics and others, either alone or in calculations, shall be used to determine whether or not a particular module meets the standards set by the user. The SQA shall also provide a summary of standards met and violated by the modules in the system being analyzed.

b. Complexity Analysis. Complexity measures shall be used to determine how simple or complicated a module is, either on an absolute scale or relative to other modules. Complexity analysis should be implemented to determine as many different complexity measures for each module analyzed as possible.

c. Quality Metrics Analysis. Quality metrics other than complexity measures shall be used to provide qualitative information about each module in the system being analyzed. These metrics may include line and comment counts as used in the software profile report, Halstead's measures, or other metrics.

3.2.1.7 Software Change Analysis

Software change analysis shall reflect the maturity and reliability of the software under analysis and should consist of these subfunctions:

a. Patch Analysis. Patch analysis shall include a comparison of master files of different program versions and shall identify where structure and data have changed and shall report the percent of change.

b. System Configuration Change Analysis. A system configuration audit shall be produced by a comparison of the system structure for all versions developed during the life cycle of the system.

3.2.1.8 Documentation Development and Review

Documentation development and review should consist of these subfunctions:

a. Comment Processing. Comment processing shall read program source code containing specifically formated comments. These comments will supply information to be used to automate documentation.

b. Program Design Language Interface. Program design language (PDL)-to-PFA interface programs should extract data from PDL reports and databases. These data will be compared to data contained in a PFA master file for the code to determine if the specification is consistent with the code and to evaluate the code and specifications for correctness and completeness.

c. User-Defined Report Generator. A user-defined report generator should be used to generate custom reports defined by the user. The report generator shall be able to read the master file or any other file used for PFA interface. A capability of reading report and data files generated by other commercially available programs should also be provided.

3.2.2 Physical Characteristics

Not applicable.

3.2.3 Reliability

PFA shall conform to the following limits on frequency and severity of software errors taken from 5.10.3.1 and 5.8.5.2 of MIL-STD-1679:

a. The number of unresolved software errors (excluding documentation errors) shall not exceed the following:

Severity	Limits
Priority 1 and 2 (high)	Zero
Priority 3 (medium)	One per 70K of machine instruction words or fraction thereof.
Priority 4 and 5 (low)	One per 35K of machine instruction words or fraction thereof.

b. Intermittent errors shall be included in the count of software errors and receive no special consideration.

c. The number of unresolved technical errors in all of the deliverable documentation shall not exceed the sum of three, plus one for every 25K of machine instructions or fraction thereof. For example, for a program having 300K machine instructions: 3 + 12 = 15 allowable documentation errors.

d. All software errors discovered during the software quality test shall be documented.

e. The following is an explanation of the levels of software error severity:

Priority 1 - An error which prevents the accomplishment of an operational or mission essential function in accordance with official requirements (e.g., causes a program stop), which interferes with an operator or mission essential function, or which jeopardizes personnel safety.

Priority 2 - An error which adversely affects the accomplishment of an operational or mission essential function in accordance with official requirements so as to degrade performance and for which no alternative work-around solution exists; or which interferes with an operator to the extent that the operator adversely affects the accomplishment of an operational or mission essential function so as to degrade performance and for which no alternative work-around solution exists. (Reloading or restarting the program is not an acceptable work-around solution.)

Priority 3 - An error which adversely affects the accomplishment of an operational or mission essential function in accordance with official requirements so as to degrade performance and for which there is a reasonable alternative work-around solution; or which interferes with an operator to the extent that the operator adversely affects the accomplishment of an operational or mission essential function so as to degrade performance and for which there is a reasonable alternative work-around solution. (Reloading or restarting the program is not an acceptable work-around solution).

Priority 4 - An error which is an operator inconvenience or annoyance and does not affect a required operational or mission essential function.

Priority 5 - All other errors.

3.2.4 Maintainability

PFA shall be maintained and updated on the basis of new requirements not outlined in this specification. Each program change shall include a test and verification phase along with the appropriate documentation changes. Quantitative estimates shall be given for the following maintenance considerations:

a. Time and level of technical skill needed to design a functional module.

b. The number and extent of impacts caused to existing PFA subprograms by the substitution of a new subprogram for a pre-existent one and by the integration of an entirely new subprogram into the PFA.

3.2.5 Transportability

The PFA shall be designed as a general-purpose software tool, capable of being transported to other computer systems and of being maintained for future software evaluation requirements of tactical command and control communication electronic computer systems. The software shall operate on commercially available hardware and operating system software located at other government installations.

3.3 Design and Construction

3.3.1 Workmanship

The PFA shall be a software system whose functions are composed of computer programs. The computer programs shall be designed in a modular, hierarchically structured manner. The design shall provide the flexibility for transporting the program to other machines and a growth potential for expansion of additional software test capabilities that can accommodate future tactical computer systems. The coding shall be performed in such a manner as to ensure that the programs can be easily read, understood, tested, and maintained. The program source code listings shall contain sufficient documentation to provide meaningful explanations of the processing performed by each function. The design of the software shall be implemented in functional modules which, if modified, will have little to no impact on other modules. Each program shall run independently of the others.

3.3.2 Human Performance/Human Engineering

PFA shall have ergonomically optimized display format and shall allow easy and noncritical recovery from operator input errors.

3.4 Documentation

The following paragraphs describe the required documentation that shall be delivered with the PFA.

3.4.1 Specifications

Appropriate specifications shall be developed under the guidelines of MIL-STD-1679.

3.4.2 Supporting Operational Documentation

Supporting documentation for the operation of PFA shall include a maintenance manual and user's manual, containing descriptions of the software system and functional details on control and operations for each subsystem.

3.5 Logistics

3.5.1 Facilities and Facility Equipment

The PFA shall operate on the resources provided by the USAEPG system control facility.

3.6 Personnel and Training

3.6.1 Personnel

PFA shall require a programmer familiar with the use of the VAX 11/780 VMS operating system command directives and a working knowledge of the system editor in an interactive processing mode.

3.6.2 Training

On-the-job training for the operation of PFA shall be handled on an interactive basis on the VAX 11/780 interactive terminal, following guidelines in the PFA user's manual.

3.7 Functional Area Characteristics

The following paragraphs specify the functional area characteristics required for each of the PFA functional areas defined in paragraph 3.1.

3.7.1 Encoding Function

The modules of the encoding function shall perform the following subfunctions:

a. Processing of input data.

b. Error processing and reporting.

c. Master file generation.

d. Listing file generation.

3.7.1.1 Processing

Processing for all subfunctions of the encoding function will be performed in interactive mode.

3.7.1.2 Outputs

The encoding function will generate the following outputs:

a. Master file.

b. Program source listing file.

3.7.2 File Management Function

File management shall be performed using the editor and file management facilities provided by the commercial operating system.

3.7.2.1 Processing

All editing and other processing shall be done in interactive mode.

3.7.2.2 Outputs

File management shall generate the following outputs:

a. Error messages.

b. Updated master file.

c. Subsets of the master file.

3.7.3 Analysis and Reporting Function

The modules for analysis and reporting shall perform the following subfunctions:

a. Program structure analysis.

b. Data flow analysis

c. Finite state analysis.

d. Error analysis.

e. Software quality analysis.

f. Program logic analysis

g. Software change analysis

h. Documentation development and review

3.7.3.1 Processing

All of the above subfunctions shall run in interactive mode.

3.7.3.2 Outputs

There shall be at least one report generated for each of the subfunctions in 3.7.3.

4. QUALITY ASSURANCE PROVISIONS

4.1 General

This section outlines the quality assurance provisions to be followed during the design, development, and verification of PFA. The verification of PFA and its functional subsystems shall be consistent with the phased delivery and implementation schedules. The paragraphs below outline the tests and the responsibility for the tests of verification as well as the criteria for quality conformance inspections.

The verification of the software program shall be accomplished through system-level testing, unit- and module-level testing, and software integration testing. A delivery acceptance demonstration, based on the system-level test, shall be conducted.

Test documentation relating to quality assurance (QA) shall be generated and maintained. Documentation shall be informal in nature, consisting of memos that document specific QA monitors' meetings, program design review, coding (implementation), walkthroughs, etc. This documentation shall be made available on request. QA information saved/stored shall include, as a minimum, the following:

a. Software problems--system integration

b. Software problems--module integration

c. Operations, user manual discrepancies

d. Project status/accountability data

The user manual for this software shall also be subject to review and approval.

Acceptance of the software specification delivery shall be based upon the satisfactory demonstration of its performance and reliability. The delivery acceptance demonstration of the system program shall consist of system-level testing designed to verify that the overall requirements of the system and its interface with its subsystems are met. Records on all software problems encountered during testings and on data relating to the resolution and repair of these problems shall be maintained. The system delivery shall not be considered accepted until all reported discrepancies have been corrected.

4.1.1 Responsibility for Tests

Unless otherwise specified in the contract or task order, the supplier shall be responsible for the performance of all inspection requirements as specified herein. Except as otherwise specified, the supplier may utilize his own facilities or any commercial computer facility acceptable to the government. The government reserves the right to perform any of the inspections set forth in the specification where such inspections are deemed necessary to assure that supplies and services conform to prescribed requirements.

4.1.2 Special Tests and Examinations

The following tests shall be performed by the contractor in the verification of PFA.

4.1.2.1 Unit- and Module-level Testing

Unit- and module-level testing shall be performed to ensure the correctness of each program module prior to software integration testing. Each unit of each module shall be tested independently to verify its compliance with the appropriate module-level requirements. This testing shall be accomplished on an informal basis, using informal test documentation. All test failures shall be documented upon occurrence. Units and modules shall be retested whenever a software correction is implemented. Complete path testing shall be made at the unit level. Unit-and module-level testing may be done concurrently with software integration testing.

4.1.2.2 Software Integration Testing

Software integration testing shall be performed to verify the performance of larger program segments and module interfaces. This testing shall be accomplished during the phased integration of the program modules. As each module is integrated into the system program, that module's interfaces with the other integrated modules shall be tested. This testing shall be accomplished informally using internal software test plans and procedures. When a test failure occurs, it shall be documented, and integration testing shall be suspended until the problem is successfully corrected.

4.1.2.3 System-level Testing

The system program shall be tested on the system level to verify that the overall requirements of the system and its subsystems are met. Records shall be maintained on all software problems encountered during testing and on the resolution of these problems.

4.2 Quality Conformance Inspections

This paragraph presents quality conformance inspection criteria to be applied during the design, development, and verification of PFA. The quality conformance inspections shall include reviews of the following categories of documentation: design documents (specifications), supporting documentation (operational guide or user manual), program listings (module source code), and status accountability reports. A System Design Review (SDR) shall be held to discuss the software requirements as presented in this specification. The delivery of both the performance and design specifications shall be followed by design reviews. An SDR shall also be held to discuss the functional design as presented in the program performance specification. A Critical Design Review (CDR) shall be held to discuss the detailed implementation-level design as presented in the program design specification. The design review shall be scheduled approximately three weeks after delivery of each specification. Additional interim design reviews may also be scheduled.

A Verification Cross-Reference Index (VCRI) (Table I) shall be used to provide for the direct identification of each Section 3 requirement and the associated method for verifying that the requirements has been satisfied. An N/A in the VCRI paragraph list indicates that there is no verifiable requirement in the Section 3 paragraph.

The quality conformance inspections of the program listings shall include several separate procedures. First, a review shall be made of the module listings and the program design specifications to verify that a module listing exists for each module identified in the program design specifications and to verify that each module for which a module listings exists was identified in the program design specification. The listings for each separately assembled or compiled module shall then be reviewed, using the following criteria:

a. The module shall contain the complete processing for that function or subfunction, to permit modification of that function/module without requiring modification of other modules.

b. The module listing shall contain a prologue comment section which identifies the module, the system, the module inputs and outputs, error conditions and error processing, and a brief description of the module's function.

c. The module listing shall contain comments to describe the inputs, processing, and outputs for each routine and subroutine.

d. The program flow within the module listing shall be consistent with the program flow defined in the program design specification.

Quality conformance inspection shall also include the review of status reports and monitoring of the status and accountability information throughout

TABLE I. VERIFICATION CROSS-REFERENCE INDEX

Paragraph No.	Paragraph Name	Verification Method	Verification Category
3	Requirements	N/A	---
3.1	System Definition	N/A	---
3.1.1	PFA General Description	D	S
3.1.2	System/Interface Diagram	D	S
3.1.2.1	External Interface	D	S
3.1.2.2	Internal Interface	D	S
3.1.2.2.1	Software Standards File	D	S
3.1.2.2.2	Master File	D	S
3.1.2.2.3	User Files	D	S
3.1.2.3	Host Operating System	N/A	---
3.1.3	Government Furnished Property (Target Hardware)	N/A	---
3.1.4	Operational and Organizational Concepts	D	S
3.2	Characteristics	D	S
3.2.1	Performance Characteristics	T	S
3.2.1.1	Program Structure Analysis	T	S

Method:
I=Inspection
D=Demonstration
T=Test and Review of Test Data
N/A=Not Applicable

Category:
S=Software Integration Testing
R=Design Review

TABLE I. VERIFICATION CROSS-REFERENCE INDEX (Continued)

Paragraph No.	Paragraph Name	Verification Method	 Category
3.2.1.2	Data Flow Analysis	T	S
3.2.1.3	Finite State Analysis	T	S
3.2.1.4	Error Analysis	T	S
3.2.1.5	Program Logic Analysis	T	S
3.2.1.6	Software Quality Analysis	T	S
3.2.1.7	Software Change Analysis	T	S
3.2.1.8	Documentation Development and Review	T	S
3.2.2	Physical Characteristics	D	S
3.2.3	Reliability	D	S
3.2.4	Maintainability	I	R
3.2.5	Availability	D	---
3.2.6	Transportability	T	S
3.3	Design and Construction	I	R
3.3.1	Workmanship	I	R
3.3.2	Human Performance/Human Engineering	T	S

Method:
I=Inspection
D=Demonstration
T=Test and Review of Test Data
N/A=Not Applicable

Category:
S=Software Integration Testing
R=Design Review

TABLE I. VERIFICATION CROSS-REFERENCE INDEX (Continued)

Paragraph No.	Paragraph Name	Verification Method	Verification Category
3.4	Documentation	I	R
3.4.1	Sperifications	I	R
3.4.2	Supporting Operational Documentation	I	R
3.5	Logistics	N/A	---
3.5.1	Facilities and Facility Equipment	D	S
3.6	Personnel Training	N/A	---
3.6.1	Personnel	D	S
3.6.2	Training	D	S
3.7	Functional Area Characteristics	T	S
3.7.1	Encoding Function	T	S
3.7.1.1	Processing	T	S
3.7.1.2	Outputs	T	S
3.7.2	File Management Function	T	S
3.7.2.1	Processing	T	S
3.7.2.2	Outputs	T	S

Method:
I=Inspection
D=Demonstration
T=Test and Review of Test Data
N/A=Not Applicable

Category:
S=Software Integration Testing
R=Design Review

TABLE I. VERIFICATION CROSS-REFERENCE INDEX (Continued)

Paragraph No.	Paragraph Name	Verification Method	 Categcry
3.7.3	Analysis and Reporting Function	T	S
3.7.3.1	Processing	T	S
3.7.3.2	Outputs	T	S

Method:
I=Inspection
D=Demonstration
T=Test and Review of Test Data
N/A=Not Applicable

Category:
S=Software Integration Testing
R=Design Review

each phase of the software development effort. As a minimum, the following status information shall be provided for each module and module subroutine defined in the program design specifications:

a. Coding effort started.

b. Coding effort completed.

c. Module-level testing started.

d. Module-level testing completed.

e. System-level testing started.

f. System-level testing completed.

A record shall also be maintained of all software problems encountered during testing and of how each was resolved. For each software specification delivery, a list of the functional capabilities provided in that specification delivery shall be established and the testing/verification status of each shall be maintained.

5. PREPARATION FOR DELIVERY

The preparation for the PFA delivery shall include the following steps:

a. Generation of source code listings of all programs being delivered.

b. Generation of a machine readable software source.

c. Updated operational guides, if required. All software deliveries shall be in accordance with the phased development and implementation schedule agreed upon.

APPENDIX F

PFA USERS MANUAL

PROGRAM FLOW ANALYZER

USERS MANUAL

JANUARY 1984

PROGRAM FLOW ANALYZER
USERS MANUAL
TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

TABLE OF CONTENTS (CONTINUED)

APPENDICES

SECTION 1. GENERAL

1.1 Purpose of the Users Manual. The objective of the Users Manual for the Program Flow Analyzer (PFA) system, TECOM Project Number 7-CO-RDO-EP1-004, is to provide non-ADP personnel with the information necessary to use the PFA system effectively.

This manual reflects the current implementation of the PFA system on the VAX computer. The capabilities described in this document represent a subset of the planned system as defined in the PFA A-level specification.

1.2 Project References. PFA is a software analysis system. It is made up of various programs which identify software quality, structure, modification, and features which aid or interfere with program maintainability and reliability.

PFA is sponsored by the US Army Electronic Proving Ground (USAEPG) at Fort Huachuca, Arizona.

The following documents are applicable:

a. USAEPG, Methodclogy Investigation Proposal--Program Flow Analyzer, March 1979.

b. USAEPG, Program Flow Analyzer A-level Specification, 4 April 1980.

c. USAEPG, Program Flow Analyzer Detailed Work Plan, 30 October 1982.

d. Leslie Claudio, Methodology Investigation Final Report--Program Flow Analyzer, 30 October 1982.

e. USAEPG, PLRS DT-II Final Report, December 1981.

f. Department of Defense, DOD-STD-7935.1-S, Automated Data Systems Documentation Standards, 13 September 1977.

g. Department of Defense (Navy), MIL-STD-1679, Weapon System Software Development, 1 December 1978.

1.3 Terms and Abbreviations. Terms, abbreviations, acronyms, and definitions are given in appendix A.

1.4 Security and Privacy. The PFA system is unclassified and is currently set up to identify all output as unclassified. To process classified information the program must be modified.

SECTION 2. SYSTEM SUMMARY

2.1 System Application. PFA is a software tool for automating software analyses which are normally performed manually. These analyses are time consuming and are often omitted.

PFA provides system structure, module structure, software quality assessment, and system-level and module-level comparison of versions of software. PFA can also be used to generate system documentation.

Document review of code and design-level specifications, software quality metrics, and comparison of different versions of software are normally performed manually. PFA generates reports that are comparable to design specifications; this reduces the effort needed to trace from code to design specifications. PFA computes and compares software quality metrics to user-selected software quality standards, thereby automating the software quality assessment. PFA compares two versions of software, in an overall comparison and then a comparison of selected modules, and generates reports which clearly locate changes.

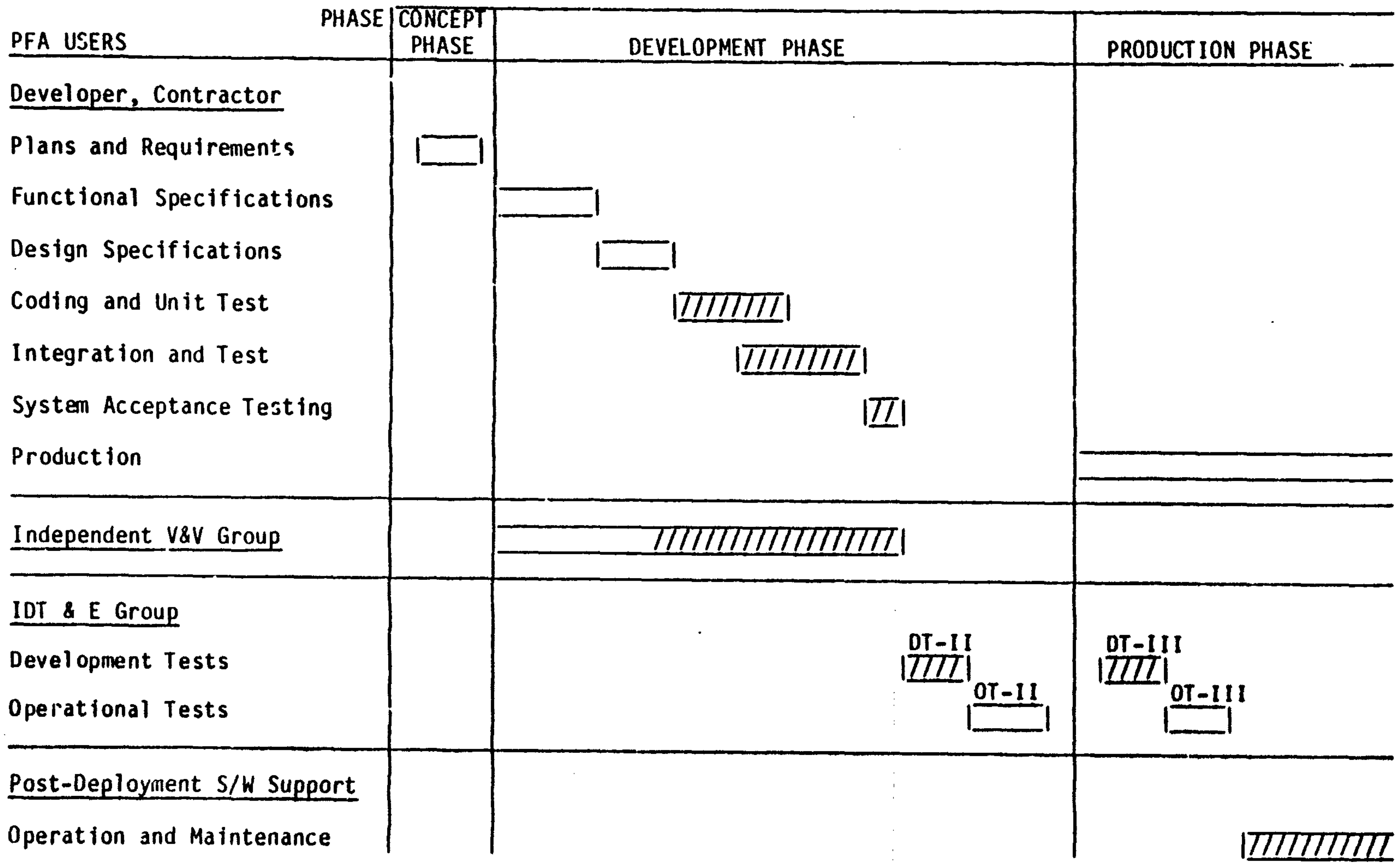
2.2 System Operation. The PFA system can be used by the developer, the independent verification and validation (V&V) contractor, the independent development test and evaluation (IDT&E) activity, the independent evaluation activity, and the post-deployment software support (PDSS) activity. Figure 2-1 shows the relationship of the various users of PFA within the acquisition life cycle. The shaded areas show when PFA can be used and who might use it.

PFA reduces the effort required to document a system. Using COMPRO or a full capability encoder with STAMP provides the developer with a procedure call structure chart and HIPO charts for each procedure. These can be used to create or update program documentation CPCIs.

PFA reduces the effort for documentation reviews. The system-specific encoder and the STAMP program provide the Independent V&V and development tester with procedure call structure charts and HIPO charts which are comparable with the developer's program documentation CPCIs. Lists of procedures called which are external to the developer's software, a list of the highest level procedures to compare against the functions described in function specifications, and an alphabetical directory of all procedures supplied by the developer are also provided.

PFA reduces the effort required to check software against standards and software quality metrics. The system-specific encoder and the SOFTPRO program provide the developer and tester with a list of procedures, pointing out the deviations from software standards and acceptable software quality measures for each procedure.

PFA reduces the effort required to monitor changes between old and new versions of software. The system-specific encoder and the SYSTRUCT and PATCHANA programs provide the developer, configuration controller, and testers with an audit of changes between two versions of the same software from the system level and the procedure level. Figure 2-2 shows an assignment of configuration control responsibilities during various periods in the acquisition cycle. Figure 2-3 shows the relationship of the various user activities to the individual programs of the PFA system.



The shaded areas indicate PFA use.

Figure 2-1. Use of PFA in the Acquisition Life Cycle

PHASE CHANGE CATEGORY	CODE AND UNIT TEST	SYSTEM INTEGRATION	ACCEPTANCE TEST	PRODUCTION
Code change	Programmer	Integration Manager	ICCB	SCCB
Design change - Intra-CPC	CPC Manager	ICCB	ICCB	SCCB
Design change - Intra-CPCI	CPCI Manager	ICCB	ICCB	SCCB
Design change - Inter-CPCI	System Software Manager	ICCB	ICCB	SCCB

ICCB - Internal Change Control Board (developer, contractor)

SCCB - System Configuration Control Board (customer, end user)

Figure 2-2. PFA Use in Support of Software Change Control and Implementation

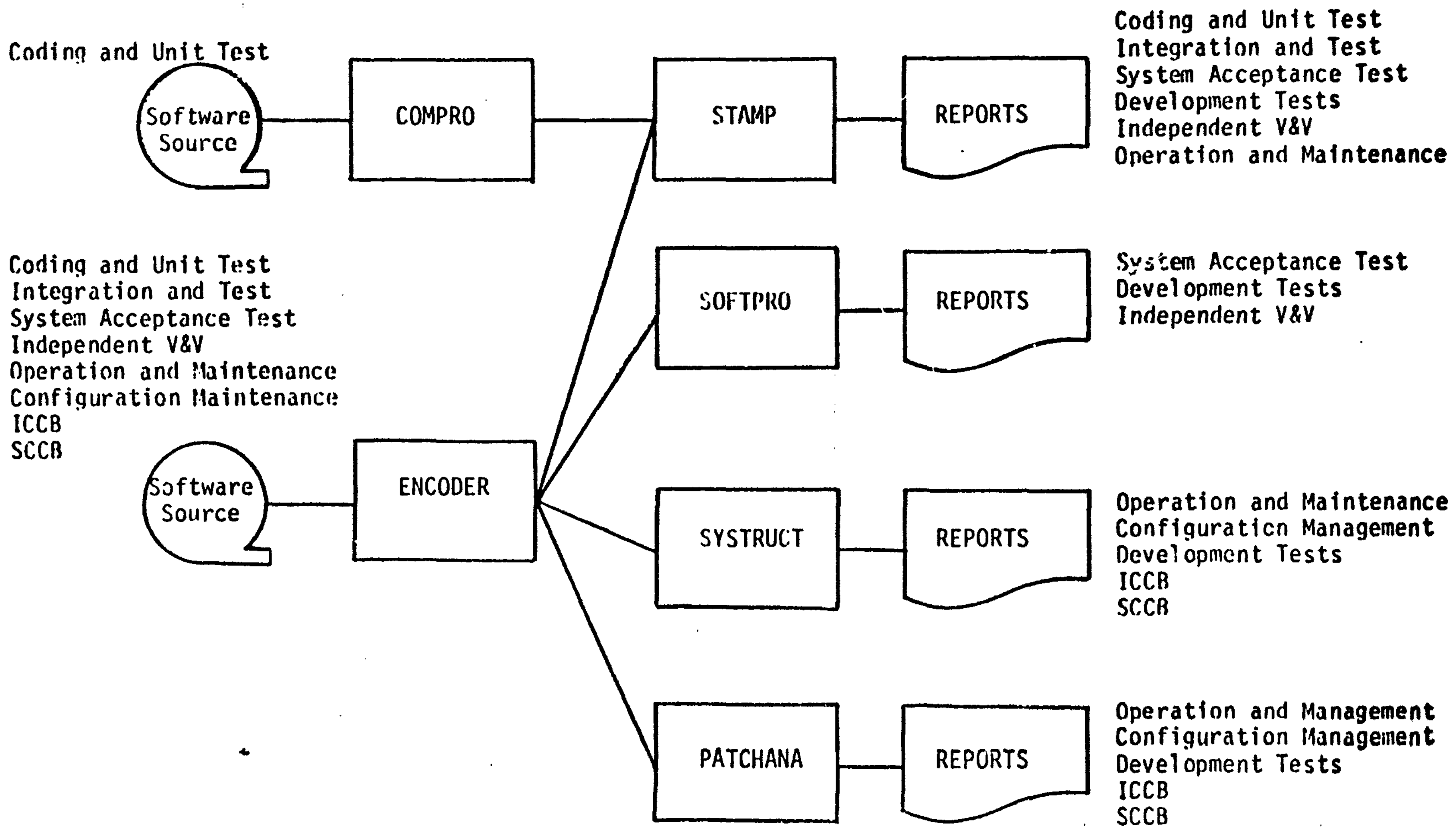


Figure 2-3. PFA End User Activities

2.3 System Configuration. PFA is currently implemented on a DEC VAX/VMS system, model 11/780, and requires a dis. and a 9-track tape drive.

2.4 System Organization. The flow of data through the system is shown in figure 2-4. A general description of the organization of the PFA system is provided below.

2.4.1 Encoder (AUTOxxxx). The encoder is a front-end program to the PFA system, customized for each application to accommodate the specific machine/language combination of the software system being analyzed, which translates computer/software/language specific programs into a representation which is stored in a data base called the master file. This master file is then read by other PFA programs that generate reports.

2.4.2 Comment Processor (COMPRO). COMPRO extracts and processes comments for creating documentation. COMPRO reads program source code files which contain comments in . specific format and creates a master file which can be processed by other PFA programs to generate reports used for program documentation. PFA programs are commented in this manner to provide automated HIPO and structure charts for the PFA system. A source listing of the input is also provided.

2.4.3 Structure, Timing, Analysis, Modeling Program (STAMP). STAMP performs program structure analysis. The STAMP program reads the master file and generates reports on the overall system under analysis to aid in modeling that system and to orovide information about the structure of that system.

2.4.4 Software Profile Program (SOFTPRO). SOFTPRO performs software quality analysis. SOFTPRO reads the master file and a software standards file and generates two reports on software standards violations.

2.4.5 System Structure Comparator (SYSTRUCT). SYSTRUCT performs analysis of system configuration changes. SYSTRUCT reads two master files representing different versions of the software system under analysis. SYSTRUCT generates a report on system-level changes in the analyzed system.

2.4.6 Patch Analysis Program (PATCHANA). PATCHANA performs software patch (source code change) analysis. PATCHANA reads two master files or subsets of master files (work files) representing two versions of the software being analyzed. Selection can be made interactively or via the selection file, which contains a list of modules to be compared. The report lists the path segment structure for both versions side by side and indicates (with codes and arrows) the location of differences and what was changed for each version.

2.5 Performance. PFA supports various software evaluation activities by reducing the effort required to perform those activities manually or by making performance of those activities feasible, given constraints on time and effort. PFA reduces effort by presenting its output in a format which is comparable to the CPCI for the program under analysis.

To process a master file containing about 200 modules, the STAMP program takes one hour, SOFTPRO takes about 15 minutes, and SYSTRUCT takes 30 minutes. PATCHANA takes about two minutes per module to be compared if full master files are used, less if work files are used.

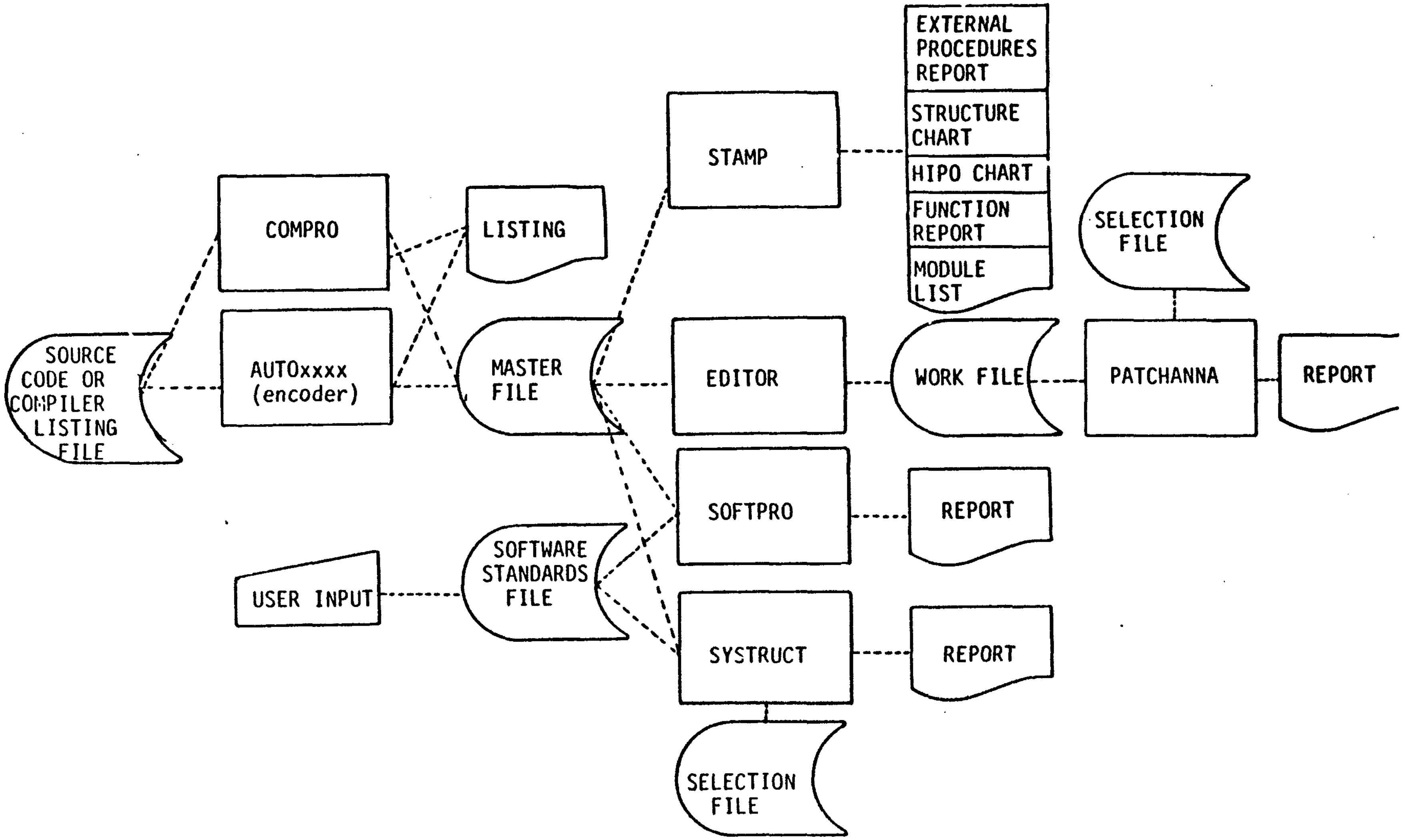


Figure 2-4. Structure of the PFA System

2.6 Data Bases. PFA uses two data base files, the master file and the standards file. The master file contains summary information on the program to be analyzed. It is used by STAMP, SOFTPRO, SYSTRUCT, and PATCHANA to analyze and report on program structure, maintainability, and software quality.

The standards file contains information on software standards. It is used by SOFTPRO to calculate and report standards violations by the program being analyzed and by SYSTRUCT to identify software quality metrics being compared.

2.7 General Description of Inputs, Processing, Outputs. Following is a general description of PFA inputs, data flow, and resultant outputs.

2.7.1 Inputs. Table 2-I lists inputs for the PFA programs.

2.7.1.1 Source Files. The only PFA programs to use software source files are COMPRO and AUTOxxxx (system-specific encoders).

2.7.1.2 Standards File. The standards file is used by SOFTPRO and SYSTRUCT to calculate and report software standards violations. The standards file describes various software standards, including such information as description, range to be within standard, expression for calculation, Military Standard Reference, and type of standard (code or documentation). The standards file is created in two parts. The first part is created when the encoder is written. It contains the descriptions of the basic standards, those standards whose values are calculated by the encoder and output in the master file. The second part of the standards file is created by the user any time before PFA run time. It contains the user-defined standards, those standards which are calculated in terms of the basic standards or other user-defined standards. These two parts exist in the same file and may be intermixed. The name of the standards file to be used is specified by the user, but by convention has the extension STD.

2.7.1.3 Master File. The master file, or a subset of it, is used by STAMP, SOFTPRO, SYSTRUCT, and PATCHANA to analyze and report on program structure, maintainability, software quality and source code changes. The master file contains summary information on the program being analyzed. This information includes the module names, prologues, flow of control, subroutine calls, variable usage, comments, entry points, and values of varying software qualities. The master file is output by an encoder or COMPRO. The name of the master file to be used is specified by the user, but by convention has the extension MST.

2.7.1.4 Selection File. The selection file contains the names of the modules in the master file to be processed by PATCHANA. It is optional, as the module names may also be specified at run time. The selection file may have been prepared manually by the user, or it may have been output by SYSTRUCT. The selection file output by SYSTRUCT contains the names of the modules determined to have probable code-level changes.

2.7.1.5 Terminal Inputs. The terminal inputs are those inputs which the user may provide at the console. Terminal inputs are required at the beginning of STAMP, SOFTPRO, PATCHANA, SYSTRUCT, and COMPRO runs. The program prompts for user responses such as input file names, output file names, and specification of desired reports and program options.

TABLE 2-I. INPUTS AND OUTPUTS OF PFA PROGRAMS

PROGRAM	INPUTS	OUTPUTS
COMPRO	PFA Source Listings	Master File Encoder Listing
STAMP	Master File	External Procedures Report Function List Structure Charts Module List HIPO Charts
SOFTPRO	Master File Standards File	Software Standards Violations Details Report Software Standards Violations Summary Report
SYSTRUCT	Master File Standards File	Report Selection File
PATCHANA	Two Master Files or Subsets Selection File (Optional)	Report

2.7.2 Processing. The flow of data between PFA programs is shown in figure 2-4. In addition, the terminal inputs to STAMP, SOFTPRO, SYSTRUCT, and PATCHANA are used to select the analysis options and the reports to be output.

2.7.2.1 Encoder (AUTOxxxx). The encoder is a front-end program to the PFA system which translates computer/software/language-specific programs into a representation which is stored in a data base called the master file. This master file is then read by other PFA programs which generate reports. The encoder is customized for each application to accommodate the specific machine/language combination of the software system being analyzed.

2.7.2.2 Comment Processor (COMPRO). COMPRO extracts source code comments for creating documentation. COMPRO reads program source code files which contain comments in a specific format. The information contained in these comments is used to create a master file which can be processed by other PFA programs to generate reports used for program documentation. PFA programs are commented in this manner to provide automated HIPO and structure charts for the PFA system.

2.7.2.3 Structure, Timing, Analysis, Modeling Program (STAMP). STAMP provides the program structure analysis function. The STAMP program reads the master file and generates reports on the overall system under analysis to aid in modeling that system and to provide information about its structure. STAMP processes the master file (PFA data base) to produce the structure charts from the subroutine calls information in the master file. STAMP also extracts the module names to provide module and function lists. Additionally, the extracted function and module names are examined to determine which routines are not defined locally, and outputs these in an external procedures report.

STAMP develops a HIPO chart based upon module information. The module name, description, prologue comments, and path segment information (inputs, outputs, quantifiers, node-path labels/branches, and inline comments) are reported from the master file. Information on modules called and calling modules is derived from subroutine calls information. This process of creating the HIPO chart is then repeated for each module defined in the master file.

2.7.2.4 Software Profile Program (SOFTPRO). SOFTPRO provides the software quality analysis function. It requires as input both a master file and software standards file.

SOFTPRO takes the formulas from the standards file and data from the software quality records of the master file and computes the values of the software quality parameters. It then takes the ranges from the standards file and determines which parameters are within allowable ranges. The results of the computations are passed to the output section to generate the exception, summary and other reports.

2.7.2.5 System Structures Comparator (SYSTRUCT). SYSTRUCT provides analysis of system configuration changes. SYSTRUCT reads two master files representing different versions of the software system under analysis and generates a report on system-level changes. These changes, determined at the module level, are derived by examining differences in subroutine calls, number of paths, and the software quality metrics between the two versions. This information is then formatted into a report for each module which displays both the old and new information and an indication of which items changed.

2.7.2.6 Patch Analysis Program (PATCHANA). PATCHANA provides a software patch (source code change) analysis function. PATCHANA reads two master files or subsets of master files representing two versions of the software being analyzed. A selection file which contains a list of modules to be compared is also read.

The master files are examined to locate information on the modules listed in the selection file. Path segment information, for the old and new modules, is processed to identify differences in structure, subroutine calls, variable usage, and the three quantifiers. This information is presented in a report which lists both versions of the module and flags the differences.

2.7.3 Outputs. Table 2-I lists PFA program outputs.

2.7.3.1 AUTOxxxx and COMPRO. AUTOxxxx and COMPRO output master files and reformatted source listings. COMPRO's master file is based on specially formatted comments in the input source listings.

2.7.3.2 STAMP. STAMP provides five reports, as follows:

a. The external procedures report lists each module which calls subroutines outside the set of modules being analyzed and the name of the subroutines called.

b. The structure chart shows the module call hierarchy of the system and flags external and recursive subroutines.

c. The function list provides a list of all task-level modules in the system, with the name of the module, a brief description, and version information.

d. The module list contains a directory of the modules being analyzed, arranged in alphabetical order. It also contains an indicator to signal whether the module is task level, subroutine level or an unused entry point; the top and bottom hierarchical level on which the module is called; a brief module description; the version information; and the page number of the source listing where the module is located.

e. The HIPO chart lists the name of the module, a brief description, the version information, and the page number of the source description. The HIPO chart also includes the path segment structure of the module. Each path segment contains the names of modules called, comments about the path, three quantifiers (discussed below) and the sequence number range of the source statements which make up the path segment. The path segment information includes inputs and outputs for the segment.

The quantifiers are user-selected values (determined by the particular encoder used) extracted from the software being analyzed. Examples are execution module timing, source statement counts, and machine instruction counts and are determined by encoder design. Those items which are not available do not prohibit processing. The STAMP program will report whatever information is available.

2.7.3.3 SOFTPRO. SOFTPRO provides two reports, as follows:

a. The software profile reports for each module list the violations of software standards as defined by the user in the software standards file. The violations state the standard, the value for the software being analyzed, the permitted range of values, and the reference to the regulation from which the standard is derived.

b. The profile summary contains the list of standards, the number and percent of modules meeting each standard, the number and percent of modules violating each standard, and the reference to the regulations containing each standard. Additionally, the quality values are summarized for all the modules and listed individually for each module.

2.7.3.4 SYSTRUCT. The SYSTRUCT report lists changes in variables used by each module, changes in software quality parameters (the ones reported by SOFTPRO), and modules which have been added to or deleted from the current version of the software being analyzed. The changes are of two types: those affecting executable code and those affecting code documentation. When changes affect executable code, a list of changed modules is generated for use by the patch analysis program.

2.7.3.5 PATCHANA. The PATCHANA report lists the path segment structure for both versions side by side and indicates (with codes and arrows) the location of differences and what was changed for each version. When structural differences occur, the differences are flagged from their start either through the remainder of the module or to the point where the structures of the two versions are congruent. PATCHANA finds differences in structure, subroutine calls, variable usage, and the three quantifiers.

SECTION 3. STAFF FUNCTIONS RELATED TO TECHNICAL OPERATIONS

3.1 Initiation Procedures. The first step for using PFA for software analysis is to obtain the encoder, with documentation, for the particular programming language and computer system for which the software being analyzed is written.

The second step is to obtain the source code or compile listing for the software under analysis on machine-readable form (tape or disk).

Next a file containing software standards to be applied to the software under analysis must be prepared, and the encoder must be run to create the master file.

The system manuals for the particular computer system where the PFA system is being run should provide specific log-on and program running procedures.

3.2 Input Requirements. The user must obtain the source code for the system being analyzed, in the format required by the encoder being used, or the source code must be prepared in correct format for COMPRO if PFA is being used for documentation.

A standards file must be prepared in a format prescribed by this document and in accordance with the encoder design.

The user must answer the prompts output by the PFA programs, which request the names of data and report files and processing options.

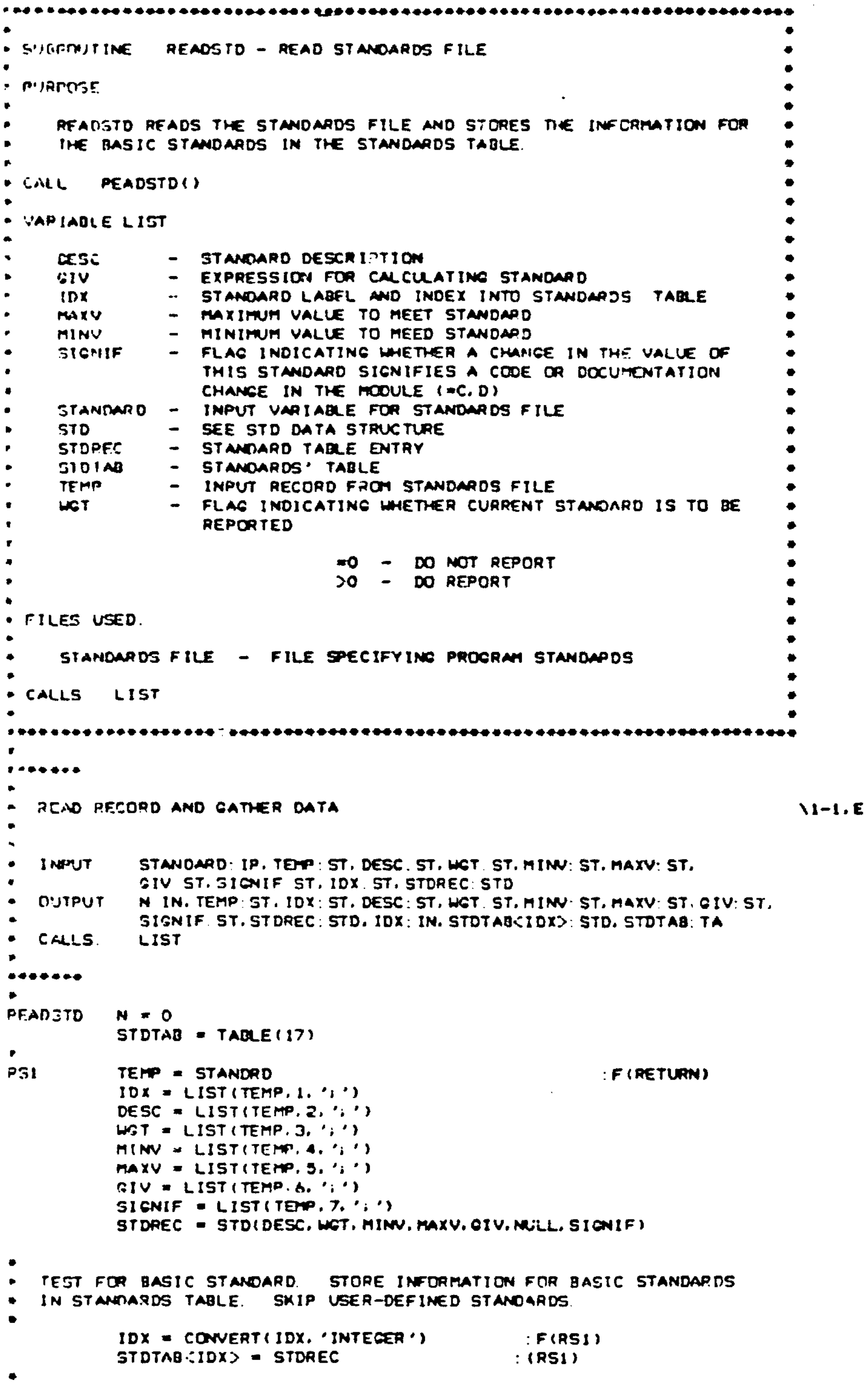
A file containing machine/language-specific source code or compiler listings is converted to a common format, independent of the machine/language combination, creating a master file used by the generic report programs of the PFA system.

When COMPRO is used to extract information for documentation, the input consists of specially formatted comment statements. These format statements may be intermixed with executable statements, as encountered in a normal program, though the latter are not essential. Input which consists entirely of the special comment statements is similar to a program design language.

System-specific encoders (AUTOxxxx) require input which is dependent on the specific encoder design. This input would normally consist of the software source code or a compiler listing of the source. Encoders may require that certain compiler options be exercised in creating the listing file (e.g., specifying the option of generating an assembler-level listing).

3.2.1 Source File. Figure 3-1 is an example of a specially commented PFA source module. The only source file documented in this manual is the PFA source used by COMPRO to produce a master file. COMPRO processes specially formatted comments in source code to produce the master file.

3.2.2 Standards File. Each record in the standards file consists of eight variable length fields separated by semicolons. Figure 3-2 is a sample standards file. The standards file records and fields are described in appendix B.



```
************************************************************************
*                                                                      *
* SUBROUTINE   READSTD - READ STANDARDS FILE                           *
*                                                                      *
* PURPOSE                                                              *
*                                                                      *
*    READSTD READS THE STANDARDS FILE AND STORES THE INFORMATION FOR   *
*    THE BASIC STANDARDS IN THE STANDARDS TABLE.                       *
*                                                                      *
* CALL   READSTD()                                                     *
*                                                                      *
* VARIABLE LIST                                                        *
*                                                                      *
*    DESC      -  STANDARD DESCRIPTION                                 *
*    GIV       -  EXPRESSION FOR CALCULATING STANDARD                  *
*    IDX       -  STANDARD LABEL AND INDEX INTO STANDARDS  TABLE       *
*    MAXV      -  MAXIMUM VALUE TO MEET STANDARD                       *
*    MINV      -  MINIMUM VALUE TO MEED STANDARD                       *
*    SIGNIF    -  FLAG INDICATING WHETHER A CHANGE IN THE VALUE OF     *
*                 THIS STANDARD SIGNIFIES A CODE OR DOCUMENTATION      *
*                 CHANGE IN THE MODULE (=C,D)                          *
*    STANDARD  -  INPUT VARIABLE FOR STANDARDS FILE                    *
*    STD       -  SEE STD DATA STRUCTURE                               *
*    STDREC    -  STANDARD TABLE ENTRY                                 *
*    STDTAB    -  STANDARDS' TABLE                                     *
*    TEMP      -  INPUT RECORD FROM STANDARDS FILE                     *
*    WGT       -  FLAG INDICATING WHETHER CURRENT STANDARD IS TO BE    *
*                 REPORTED                                             *
*                                                                      *
*                               =0  -  DO NOT REPORT                   *
*                               >0  -  DO REPORT                       *
*                                                                      *
* FILES USED.                                                          *
*                                                                      *
*    STANDARDS FILE  -  FILE SPECIFYING PROGRAM STANDARDS              *
*                                                                      *
* CALLS   LIST                                                         *
*                                                                      *
************************************************************************
*
*******
*
*  READ RECORD AND GATHER DATA                                                 \1-1.E
*
*
*  INPUT     STANDARD: IP, TEMP: ST, DESC: ST, WGT: ST, MINV: ST, MAXV: ST,
*            GIV: ST, SIGNIF: ST, IDX: ST, STDREC: STD
*  OUTPUT    N: IN, TEMP: ST, IDX: ST, DESC: ST, WGT: ST, MINV: ST, MAXV: ST, GIV: ST,
*            SIGNIF: ST, STDREC: STD, IDX: IN, STDTAB<IDX>: STD, STDTAB: TA
*  CALLS.    LIST
*
*******
*
READSTD   N = 0
          STDTAB = TABLE(17)
*
RS1       TEMP = STANDRD                                 :F(RETURN)
          IDX = LIST(TEMP,1,';')
          DESC = LIST(TEMP,2,';')
          WGT = LIST(TEMP,3,';')
          MINV = LIST(TEMP,4,';')
          MAXV = LIST(TEMP,5,';')
          GIV = LIST(TEMP,6,';')
          SIGNIF = LIST(TEMP,7,';')
          STDREC = STD(DESC,WGT,MINV,MAXV,GIV,NULL,SIGNIF)

*
*  TEST FOR BASIC STANDARD.  STORE INFORMATION FOR BASIC STANDARDS
*  IN STANDARDS TABLE.  SKIP USER-DEFINED STANDARDS.
*
          IDX = CONVERT(IDX,'INTEGER')         :F(RS1)
          STDTAB<IDX> = STDREC                 :(RS1)
*
```

Figure 3-1. A Sample PFA-Commented Source Program

```
1;COMMENT LINES IN PROLOG > 30;2;30;9999;;D;MIL-STD-1679, 5.4 4.1;
A;COMMENT LINES IN PROLOG > 60;2;60;9999;(1),;D,EPG, DOCUMENTATION;
B,COMMENT LINES IN PROLOG > 120;2;120;9999;(1),;D;EPG, DOCUMENTATION,
2;NUMBER OF NON-BLANK COMMENT LINES;0;0;9999;;D;EPG, DOCUMENTATION;
3,NUMBER OF NON-COMMENT LINES;0;0;9999;;D;;
4;NUMBER OF EXECUTABLE STATEMENTS < 200;2;0;200;;D;MIL-STD-1679, 5.3.7;
5;NUMBER OF STATEMENTS WITH EMBEDDED COMMENTS;0;0;9999;;D;EPG, DOCUMENTATION;
C;50% CODE COMMENTED;2;50;99999;(2),(5),+,100,*,(4),/,;D,EPG, DOCUMENTATION,
D,75% CODE COMMENTED;2;75;99999;(2),(5),+,100,*,(4),/,;D;EPG, DOCUMENTATION,
E,100% CODE COMMENTED;2;100,99999;(2),(5),+,100,*,(4),/,;D;MILSTD-1679,5.4 4 2;
6,NUMBER OF ENTRY POINTS = 1;1;0;1;;C;MIL-STD-1679, 5.3.3,
7;HALLSTEADS COMPLEXITY < 5;0;1;5;;C;EPG, COMPLEXITY;
8,KNOTS COMPLEXITY < 3;0;0;3;;S;EPG, STRUCTURE;
9,MCCABE'S COMPLEXITY < 10;1;1;9;;C;EPG, COMPLEXITY;
F,MCCABE'S COMPLEXITY < 20,1;1;19;(9),;C;EPG, COMPLEXITY;
G,MCCABE'S COMPLEXITY < 30;1;1;29;(9),;C;EPG, COMPLEXITY;
H;NUMBER CF LINES < 500;2;1;500;(1),(2),+,(3),+,;C;EPG, COMPLEXITY;
I;NUMBER OF DECLARATION STATEMENTS < 25;1;0;25,(3),(4),-,;C;EPG, COMPLEXITY;
J;NUMBER OF DECLARATION STATEMENTS < 50;1;0;50,(3),(4),-,;C;EPG, COMFLEXITY,
K;NUMBER OF DECLARATION STATEMENTS <100;1;0;100;(3),(4),-,;C;EPG, COMPLEXITY;
10;FUTURE DEVELOPMENT (TOP ' )WN STRUCTURE);0;1;3;;S;EPG, STRUCTURE;
11;FUTURE DEV.;0;1;100;;S;EPG, NEW METRIC;
```

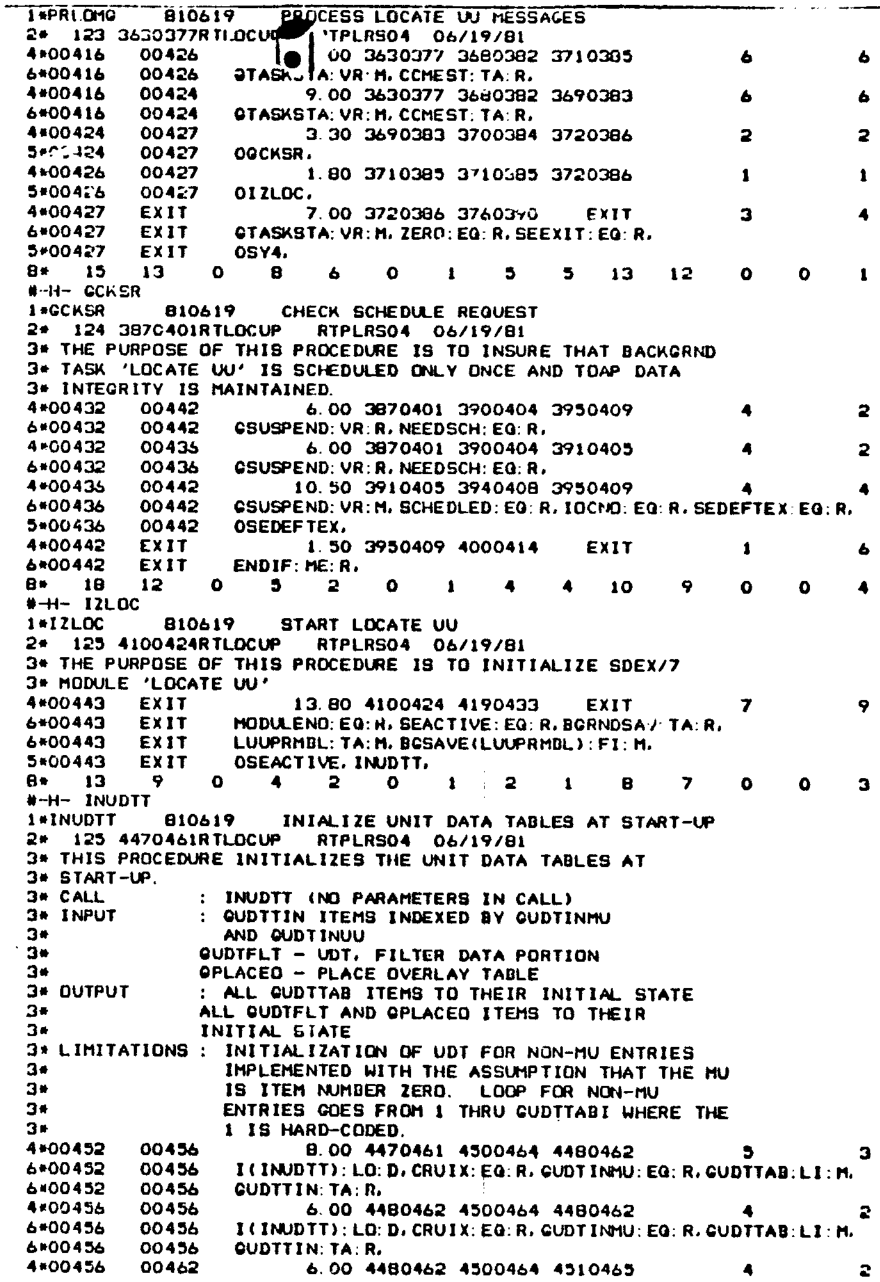
Figure 3-2. Standards File

The following describes the appropriate vocabulary for each field of a standards file record. No field may contain a semicolon.

Field	Description
Identifier	Integer for basic standards (1-14) Any other value for user-defined standards
Standards	Any text
Weighting Factor	Ø for no report on standard 1 for report on standard
Lowest Permitted Value	Any value
Highest Permitted Value	Any value (must be greater than lowest permitted value)
Calculation	Null for basic standards. A reverse Polish expression for user-defined standards. The expression may consist of constants, the operators '+', '-', '*', and '/', and the values of previously defined standards, referenced by the standard's ID in parentheses. For example, '(5),(1),/,' is a legal expression.
Code/Documentation Indicator	C for code standards D for documentation standards
Standard Reference	Any text

3.2.3 Master File. Figure 3-3 is an example portion of a master file. The master file contains summary information on the modules of the program being analyzed. Currently there are eleven record types in the file, each describing one aspect of a module. These records are grouped within the master file according to the module they describe. The aspect of a module described by each of the record types is as follows:

Type	Information
-	Header Record
0	Library Routine Definition
1	Task or Subroutine Definition
2	Listing and Version Information
3	Prologue Comments
4	Path Definition
5	Subroutine Calls Information
6	Variable Information
7	In-Line Comments
8	Software Quality Information
9	Entry Point Definition



```
1*PRLOMG     810619     PROCESS LOCATE UU MESSAGES
2*  123 3630377RTLOCUP   RTPLRS04  06/19/81
4*00416   00426              [illegible]00 3630377 3680382 3710385         6         6
6*00416   00426   GTASKSTA:VR:M,CCMEST:TA:R.
4*00416   00424              9.00 3630377 3680382 3690383         6         6
6*00416   00424   GTASKSTA:VR:M,CCMEST:TA:R.
4*00424   00427              3.30 3690383 3700384 3720386         2         2
5*00424   00427   OGCKSR.
4*00426   00427              1.80 3710385 3710385 3720386         1         1
5*00426   00427   OIZLOC.
4*00427   EXIT               7.00 3720386 3760390     EXIT        3         4
6*00427   EXIT    GTASKSTA:VR:M,ZERO:EQ:R,SEEXIT:EQ:R.
5*00427   EXIT    OSY4.
8*   15   13    0    8    6    0    1    5    5   13   12    0    0    1
#-H- GCKSR
1*GCKSR      810619     CHECK SCHEDULE REQUEST
2*  124 387C401RTLOCUP   RTPLRS04  06/19/81
3* THE PURPOSE OF THIS PROCEDURE IS TO INSURE THAT BACKGRND
3* TASK 'LOCATE UU' IS SCHEDULED ONLY ONCE AND TOAP DATA
3* INTEGRITY IS MAINTAINED.
4*00432   00442              6.00 3870401 3900404 3950409         4         2
6*00432   00442   GSUSPEND:VR:R,NEEDSCH:EQ:R.
4*00432   00435              6.00 3870401 3900404 3910405         4         2
6*00432   00436   GSUSPEND:VR:R,NEEDSCH:EQ:R.
4*00435   00442             10.50 3910405 3940408 3950409         4         4
6*00436   00442   GSUSPEND:VR:M,SCHEDLED:EQ:R,IOCMD:EQ:R,SEDEFTEX:EQ:R.
5*00436   00442   OSEDEFTEX.
4*00442   EXIT               1.50 3950409 4000414     EXIT        1         6
6*00442   EXIT    ENDIF:ME:R.
8*   18   12    0    5    2    0    1    4    4   10    9    0    0    4
#-H- IZLOC
1*IZLOC      810619     START LOCATE UU
2*  125 4100424RTLOCUP   RTPLRS04  06/19/81
3* THE PURPOSE OF THIS PROCEDURE IS TO INITIALIZE SDEX/7
3* MODULE 'LOCATE UU'
4*00443   EXIT              13.80 4100424 4190433     EXIT        7         9
6*00443   EXIT    MODULENO:EQ:H,SEACTIVE:EQ:R,BGRNDSAV:TA:R.
6*00443   EXIT    LUUPRMBL:TA:M,BGSAVE(LUUPRMBL):FI:M.
5*00443   EXIT    OSEACTIVE,INUDTT.
8*   13    9    0    4    2    0    1    2    1    8    7    0    0    3
#-H- INUDTT
1*INUDTT     810619     INIALIZE UNIT DATA TABLES AT START-UP
2*  125 4470461RTLOCUP   RTPLRS04  06/19/81
3* THIS PROCEDURE INITIALIZES THE UNIT DATA TABLES AT
3* START-UP.
3* CALL          : INUDTT (NO PARAMETERS IN CALL)
3* INPUT         : QUDTTIN ITEMS INDEXED BY QUDTINMU
3*                 AND QUDTINUU
3*               QUDTFLT - UDT, FILTER DATA PORTION
3*               QPLACEO - PLACE OVERLAY TABLE
3* OUTPUT        : ALL QUDTTAB ITEMS TO THEIR INITIAL STATE
3*               ALL QUDTFLT AND QPLACEO ITEMS TO THEIR
3*               INITIAL STATE
3* LIMITATIONS : INITIALIZATION OF UDT FOR NON-MU ENTRIES
3*                 IMPLEMENTED WITH THE ASSUMPTION THAT THE MU
3*                 IS ITEM NUMBER ZERO.  LOOP FOR NON-MU
3*                 ENTRIES GOES FROM 1 THRU QUDTTABI WHERE THE
3*                 1 IS HARD-CODED.
4*00452   00456              8.00 4470461 4500464 4480462         5         3
6*00452   00456    I(INUDTT):LO:D,CRUIX:EQ:R,GUDTINMU:EQ:R,GUDTTAB:LI:M.
6*00452   00456    GUDTTIN:TA:R.
4*00456   00456              6.00 4480462 4500464 4480462         4         2
6*00456   00456    I(INUDTT):LO:D,CRUIX:EQ:R,GUDTINMU:EQ:R,GUDTTAB:LI:M.
6*00456   00456    QUDTTIN:TA:R.
4*00456   00462              6.00 4480462 4500464 4510465         4         2
```

Figure 3-3. Master File

The header record has the form "#-H-modname", left-justified. The record fields and their layouts are described in appendix B.

3.2.4 Selection File. The selection file is a list of modules to be processed by PATCHANA. Each line contains one module name, in the format required by the system on which PFA is run. The selection file cannot contain any blank records.

3.2.5 Terminal Inputs. All terminal inputs have a maximum length of 80 characters and must be left-justified.

a. Option Specifications. The affirmative response to prompts for all option requests is 'YES' or 'Y'. All other responses will be interpreted as negative.

b. File Name Specifications. All file names must be in the format required by the system on which PFA is run. The PATCHANA request for a selection file name may be answered with the response 'TTY' as well as a file name.

c. File Name List Specification. The specification consists of a list of file names, each in the format specified by b. above, separated by commas or blanks.

d. STAMP Report Specification. The specification consists of a string of contiguous letters as described below.

Letter	Requested Report
E	External Procedures Report
S	Structure Chart
F	Function List
M	Module List
H	HIPO Charts

The letters may be in any order. In addition, if no specification is made, the STAMP default is to output all five reports.

e. STAMP Variable Lists. STAMP allows the user to specify that certain variables in the program being analyzed be included or excluded from processing. The responses to the prompts for included or excluded variables must have the following format:
var1:ty1:u1,var2:ty2:u2,...varn:tyn:un,

where vari is the variable name, tyi is an abbreviation for variable type, as specified in the appropriate encoder manual, and ui is the variable usage, encoded as follows:

D	Defined
R	Referenced
T	Tested
M	Modified

A null entry in one or two fields means that these fields will not be used as selection criteria.

3.3 Program Operations. The operation of the PFA is discussed below.

3.3.1 COMPRO. COMPRO extracts and processes comments from a program source file for creating program documentation. Figure 3-4 shows the data flow between COMPRO and the rest of the PFA system.

COMPRO reads program source code files which contain comments in a specific format. The information contained in these comments is used to create a master file which can be processed by other PFA programs to generate reports used for program documentation. A listing of the source code is also produced.

3.3.1.1 COMPRO Input Requirements. The only source file documented in this manual is the PFA source code used by COMPRO to produce a master file. COMPRO processes specially formatted comments to produce the master file. The requirements for the special comments are described below.

a. Prologues.

(1) Each module has a prologue.

(2) The prologue is surrounded by asterisks. This means that the prologue must be preceded and followed by a line consisting only of a string of asterisks and that each line in the prologue must begin and end with an asterisk.

(3) Every prologue contains as the first non-blank line:

```
TASK
SUBROUTINE          :    <name> - <description>
LIBRARY ROUTINE
```

Where, if the module has multiple entry points, <name> is the name of the first entry point.

(4) Following item 3 is the module description:

```
AUTHOR:
DATE STARTED:
DATE LAST MODIFIED:
PURPOSE:  <function of module>
DESCRIPTION:  <general design description>
CALL:
PARAMETER LIST:  <parameters and descriptions>
USER-DEFINED DATA STRUCTURES:  <structures, fields, and
                                descriptions>
VARIABLE LIST:  <variables and descriptions>
FILES USED:   <file names and descriptions>
CALLS:
LIMITATIONS:
ERRORS:
```

as there are appropriate data to include.

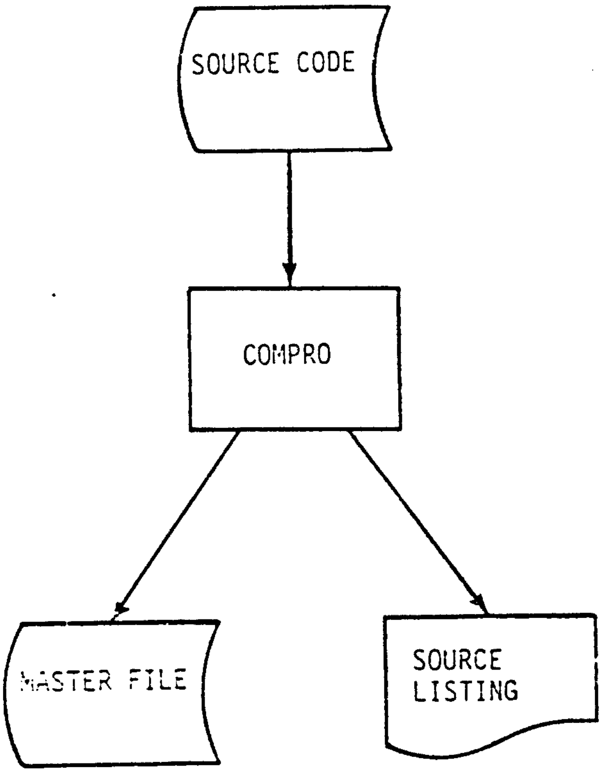


Figure 3-4. COMPRO Data Flow

(5) For each additional entry point the module contains, (4) above is followed by:

ENTRY POINT: <name> - <description>

and again, as many of the items in (4) above as appropriate.

b. In-Line Comments

(1) All in-line comments are separated from code by strings of asterisks.

(2) In-line comments do not contain ' '.

(3) The in-line comments may contain psuedo path segment specifications. These specifications document the flow of control in a module and are used by COMPRO and STAMP to create HIPO charts. If used, the path segment specifications have the following format and meaning:

(a) The specification for a path segment begins on the first comment to be included in the segment.

(b) The specification is the rightmost item on the comment line.

(c) The specification is of the form:

```
[m][n1,n2,...][E]
```

where m is the optional node label for the start of the path segment, ni are the optional node labels to which the segment branches, and E indicates branch to exit. If the *from* node label is null, the *from* node label will be assumed to be the current node count for this module. If the *to* node labels (including E) are null, the *to* node label will be assumed to be the current node count for this module plus one. If this path segment is an entry point for the module but is not the first entry point to the module, m must be the name of the entry point.

(d) The last path specification in a module has a *to* node specified.

(4) All referenced and modified variables appearing in a psuedo path segment are listed in comments within the path segment for use by COMPRO and STAMP. If there is no path segment specification, all referenced and modified variables for the module are listed. Comments listing referenced or modified variables have the respective formats:

```
INPUT:  var1[:type],var2[:type],...vari[:type],
                          .
                          .
                          .
                 ...varn[:type]
```

```
OUTPUT:  var1[:type],var2[:type],...vari[:type],
                               .
                               .
                               .
                  ...varn[:type]
```

where type is optional and is an abbreviation for the variable's type. In the case of SNOBOL the types are abbreviated as follows:

TYPE	ABBREVIATIONS
String	ST
Integer	IN
Real Number	RE
Pattern Structure	PA
Array	AR
Table	TA
Created Name	NA
Unevaluated Expression	UN
Object Code	CO
Programmer-defined	Data type name
External	EX
Input	IP
Output	OP

Several lists may be included in one path.

(5) All subroutines called in a pseudo path segment are listed in comments within the path segment for use by COMPRO and STAMP. If there is no path segment specification, all subroutines called in the module are listed. Comments listing the calls have the format:

```
calls:  call1, call2,...calli,
                 .
                 .
                 .
            ...calln
```

Several lists may be included in one path segment

3.3.1.2 COMPRO Initiation Procedures. COMPRO may be run from any interactive terminal connected to the VAX. Assuming that the user has logged onto the VAX with the correct password, he can run COMPRO by entering the command:

```
$ PFA
(respond to menu prompt for COMPRO)
```

COMPRO will then prompt the user for the following information. The user's response is underscored.

COMMAND/RESPONSE	EXPLANATION
ENTER SOURCE FILE NAME M.SNO	The entered source file name is the name of the source file to be processed by COMPRO.
ENTER LIST FILE NAME M.LST	The entered list file name is the name of the disk file containing the output source listing.
ENTER MASTER FILE NAME M.MST	The entered master file name is the name of the output master file.

COMPRO will then complete the master file generation with no further prompts.

3.3.1.3 Files Used in COMPRO Processing. COMPRO does not use temporary files.

3.3.1.4 COMPRO Recovery and Error Correction Procedures. COMPRO indicates the following errors:

a. Message: "ERR10: ERROR PROCESSING LINE IN MODULE ________"
"(line> "

This means that COMPRO has found a line in the PFA source with an unexpected format. The message indicates an error within COMPRO, and processing terminates.

b. Message: "ERR20: UNEXPECTED END OF FILE IN MODULE ________"

This means that COMPRO has found an end of file while reading a multiline list. The message indicates an error in the PFA source. Processing terminates.

In the event of a hardware error which terminates the program, the program must be restarted from the beginning. In the event of a fatal software error while running COMPRO, all files must be checked to ensure they exist and are not attached to another program; COMPRO is then rerun.

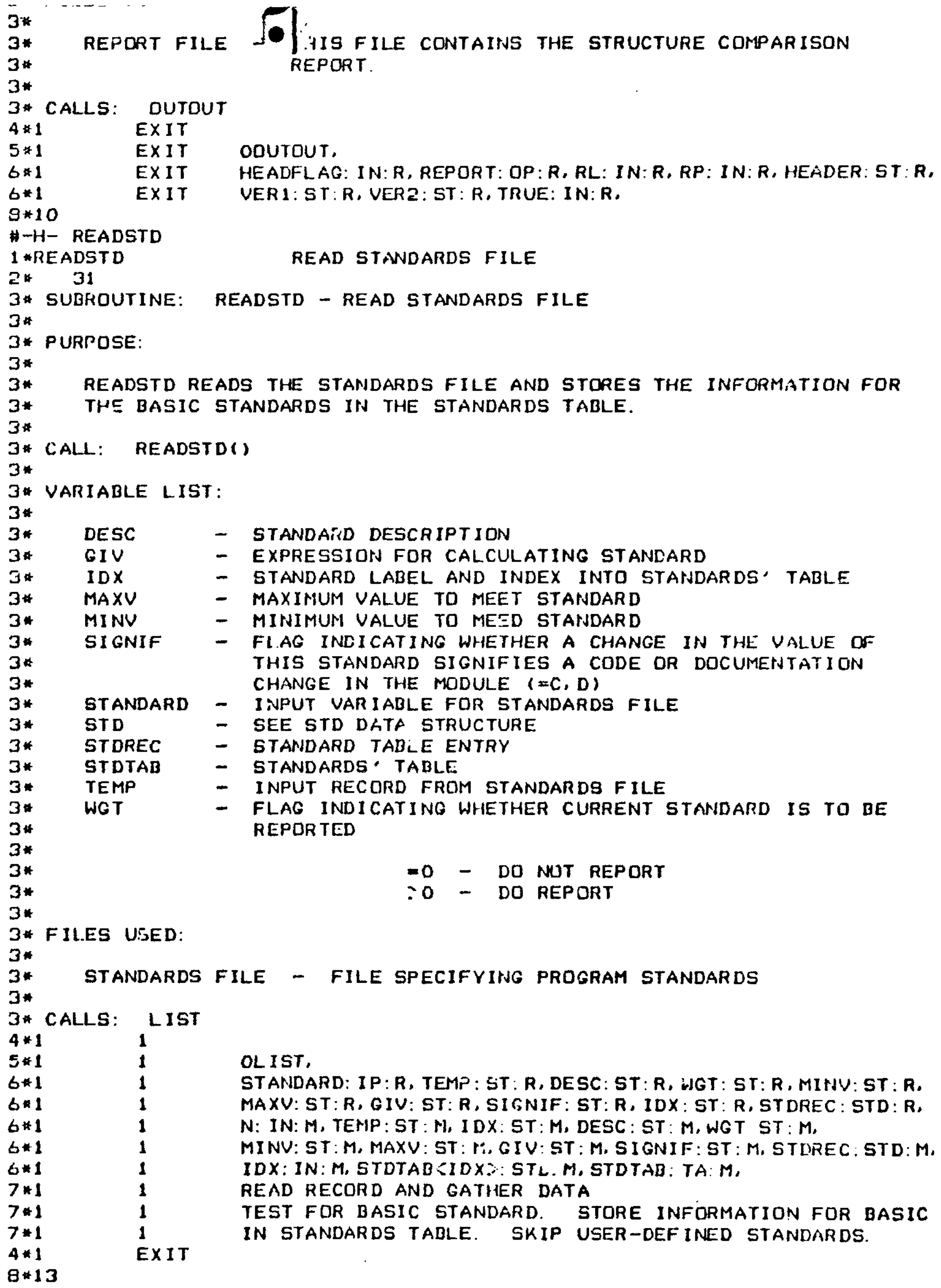
3.3.1.5 COMPRO Limitations

a. Comments in the input source listing may not contain ' '.

b. Prologues must be delimited by a string of more than 40 asterisks and other comments cannot be.

c. In-line comments must be separated from code by strings of asterisks.

3.3.1.6 COMPRO Sample Outputs. Figure 3-5 presents a sample from a master file output by COMPRO. The layout of the master file and the meaning of each field is described in appendix B. COMPRO does not output a value for the following fields described in appendix B: Source Sequence Number, Version Information, Quantifiers 1, 2 and 3, and Start, End, and Branch Sequence Numbers. In addition, COMPRO outputs only one software quality parameter to the type 8 record. This parameter is the number of non-comment lines per module.



```
3*
3*    REPORT FILE    THIS FILE CONTAINS THE STRUCTURE COMPARISON
3*                   REPORT.
3*
3* CALLS:  OUTOUT
4*1        EXIT
5*1        EXIT    OOUTOUT,
6*1        EXIT    HEADFLAG: IN: R, REPORT: OP: R, RL: IN: R, RP: IN: R, HEADER: ST: R,
6*1        EXIT    VER1: ST: R, VER2: ST: R, TRUE: IN: R,
9*10
#-H- READSTD
1*READSTD                READ STANDARDS FILE
2*   31
3* SUBROUTINE:  READSTD - READ STANDARDS FILE
3*
3* PURPOSE:
3*
3*    READSTD READS THE STANDARDS FILE AND STORES THE INFORMATION FOR
3*    THE BASIC STANDARDS IN THE STANDARDS TABLE.
3*
3* CALL:  READSTD()
3*
3* VARIABLE LIST:
3*
3*    DESC       -  STANDARD DESCRIPTION
3*    GIV        -  EXPRESSION FOR CALCULATING STANDARD
3*    IDX        -  STANDARD LABEL AND INDEX INTO STANDARDS' TABLE
3*    MAXV       -  MAXIMUM VALUE TO MEET STANDARD
3*    MINV       -  MINIMUM VALUE TO MEED STANDARD
3*    SIGNIF     -  FLAG INDICATING WHETHER A CHANGE IN THE VALUE OF
3*                  THIS STANDARD SIGNIFIES A CODE OR DOCUMENTATION
3*                  CHANGE IN THE MODULE (=C, D)
3*    STANDARD   -  INPUT VARIABLE FOR STANDARDS FILE
3*    STD        -  SEE STD DATA STRUCTURE
3*    STDREC     -  STANDARD TABLE ENTRY
3*    STDTAB     -  STANDARDS' TABLE
3*    TEMP       -  INPUT RECORD FROM STANDARDS FILE
3*    WGT        -  FLAG INDICATING WHETHER CURRENT STANDARD IS TO BE
3*                  REPORTED
3*
3*                              =0  -  DO NOT REPORT
3*                              >0  -  DO REPORT
3*
3* FILES USED:
3*
3*    STANDARDS FILE  -  FILE SPECIFYING PROGRAM STANDARDS
3*
3* CALLS:  LIST
4*1        1
5*1        1       OLIST,
6*1        1       STANDARD: IP: R, TEMP: ST: R, DESC: ST: R, WGT: ST: R, MINV: ST: R,
6*1        1       MAXV: ST: R, GIV: ST: R, SIGNIF: ST: R, IDX: ST: R, STDREC: STD: R,
6*1        1       N: IN: M, TEMP: ST: M, IDX: ST: M, DESC: ST: M, WGT ST: M,
6*1        1       MINV: ST: M, MAXV: ST: M, GIV: ST: M, SIGNIF: ST: M, STDREC: STD: M,
6*1        1       IDX: IN: M, STDTAB<IDX>: STD. M, STDTAB: TA: M,
7*1        1       READ RECORD AND GATHER DATA
7*1        1       TEST FOR BASIC STANDARD.  STORE INFORMATION FOR BASIC
7*1        1       IN STANDARDS TABLE.  SKIP USER-DEFINED STANDARDS.
4*1        EXIT
8*13
```

Figure 3-5. Master File Output by COMPRO

A reformatted source file output by COMPRO is similar to figure 3-1 but includes page numbers and page headers. The page numbers provided with this listing are referenced by some of the reports produced by STAMP, SOFTPRO, SYSTRUCT, and PATCHANA. The page headers indicate the report is unclassified.

3.3.2 STAMP. STAMP provides program structure analysis. Figure 3-6 shows the data flow between STAMP and the rest of the PFA system.

The STAMP program reads the Master File and generates reports on the overall system under analysis to aid in modeling that system and to provide information about the structure of that system. STAMP provides five reports, as follows:

External Procedure Report
Structure Chart
Function List
Module List
HIPO Diagram

3.3.2.1 STAMP Input Requirements. The encoder (AUTOxxxx) or COMPRO must have been run previously to produce the master file needed as input to STAMP.

3.3.2.2 STAMP Initiation Procedures. STAMP may be run from any interactive terminal connected to the VAX. Assuming that the user has logged onto the VAX with the correct password, he can run STAMP by entering the command:

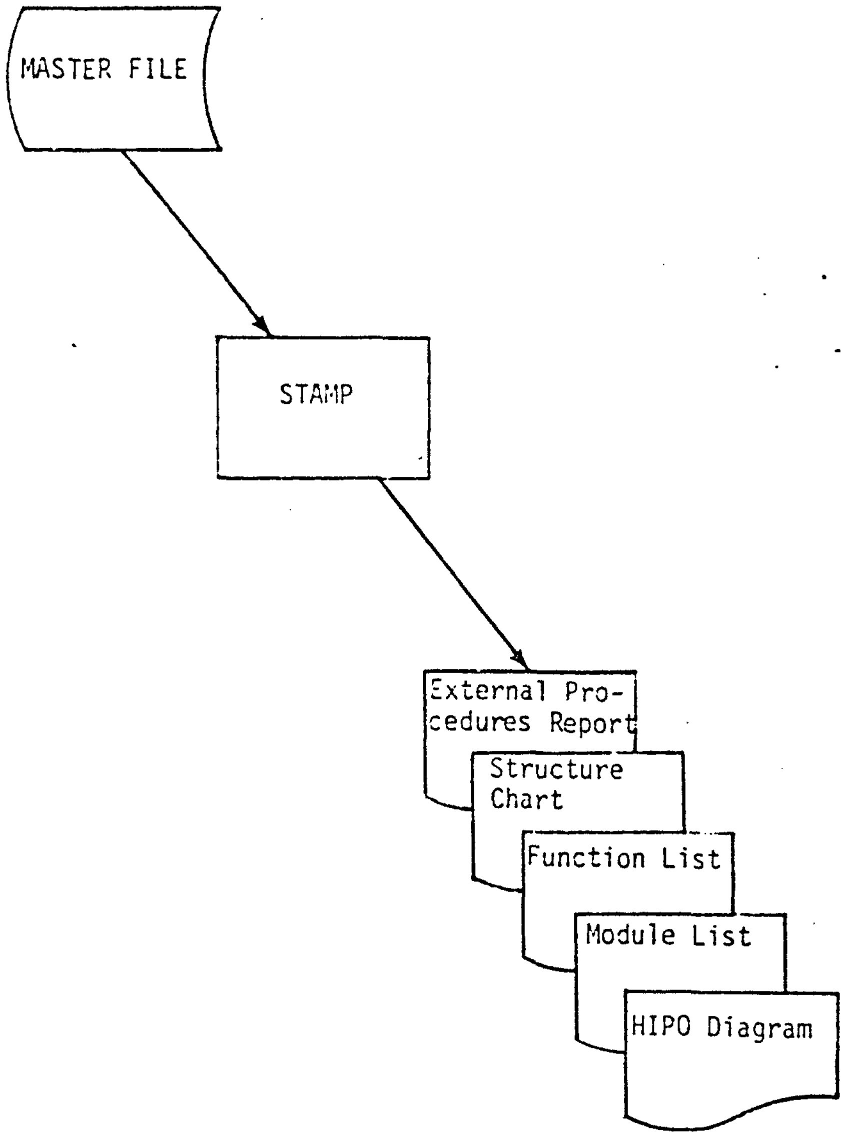


Figure 3-6. STAMP Data Flow

```
$ PFA
(respond to menu prompt for STAMP)
```

STAMP will then prompt the user for the following information:

COMMAND/RESPONSE	EXPLANATION
ENTER MASTER FILE NAME: M1.MST	This is the master file to be processed in this STAMP run.
ENTER REPORT FILE NAME: M.REP	This is the name of the disk file to contain the output STAMP reports.
DO YOU WANT ALL THE REPORTS? NO	'YES' or 'Y' sets the default to produce all of the reports, otherwise the user must select the desired reports. In the example, the negative response indicates specific reports are desired.
SELECT REPORTS (E: EXTERNAL, S: STRUCTURE, F: FUNCTION, M: MODULE, H: HIPO) SH	The request to select reports allows the user to choose the reports he desires. In the example, the structure chart and HIPO charts have been selected.
DO YOU WANT SPECIAL PROCESSING? YES	'NO' or 'N' if additional options are not desired. If 'YES', user must answer prompts to select desired processing.
DO YOU WANT TIMING MINIMIZATION? YES	Timing minimization provides information for timing analysis using the path segment descriptions in the HIPO charts. If minimization is requested, the quantifiers, consisting of such values as execution time or number of executable lines of code reported for each path segment in the HIPO chart, will show only the difference in values between paths from the same node.
DO YOU WANT PATH PRUNING? YES	Provides additional information for timing analysis using the path segment descriptions in the HIPO charts. If "YES" in addition to timing minimization, references to nodes, all of whose path quantifiers are zero, are deleted and paths reconnected as necessary.
DO YOU WANT DEBUG OUTPUT? NO	Consists of node data output to the file STAMP.TMP.
DO YOU WANT FULL STRUCTURE CHART? YES	The full structure chart includes module descriptions along with the module names.

COMMAND/RESPONSE	EXPLANATION
DO YOU WANT VARIABLE PROCESSING? YES ENTER VARIABLES TO BE INCLUDED: VAR1:IN:M,VAR1:IN:R	Variable processing is used to include or exclude specified variables from processing. The variables to be included or excluded must be specified at the next two prompts. The first prompt, for included variables, defaults to all.
ENTER VARIABLES TO BE EXCLUDED: VAR2:IN:D	The second prompt, for excluded variables, defaults to none.

The STAMP program will then complete the report generation with no further prompts.

3.3.2.3 Files Used in STAMP Processing. Two temporary files are used in STAMP processing, STAMP.TMP and STAMP1.TMP.

3.3.2.4 STAMP Recovery and Error Conditions. STAMP provides the diagnostic message: "BAD MAP FOR ____". This means that there is a *to* label without a corresponding *from* label or that the *from* or *to* label is *null* for a path datum in the master *file*. This results from *incomplete* path structure generated by the encoder or from dead code in the source code being analyzed.

In the event of a hardware error which terminates the program, the program must be restarted from the beginning.

In the event of a fatal software error while running STAMP, the user should check to make sure all files exist and are not attached to another program, and rerun STAMP.

3.3.2.5 STAMP Limitations. STAMP may cause memory thrashing if there are more than 1000 path records in any module or if there are more than 300 path records in any module and path pruning is being used.

3.3.2.6 STAMP Sample Outputs.

a. External Procedure Report (see figure 3-7). This report shows all calls to external procedure in the program being analyzed, where an external procedure is any module not appearing anywhere in the file. The calls are shown in the order in which they appear in the file.

b. Structure Chart (see figure 3-8). The structure chart shows the calling structure of the program being analyzed. Each call statement occurring anywhere in the program is shown, along with the "level" of the call. The main module is listed first, with a call level of zero. After each module, all modules called by that module are listed, at the next-highest call level.

c. Function List (see figure 3-9). The function list is a list of all task-level modules in the system. The name of the module, a description of the module, and version information are provided.

```
                                                  *******************
                                                  *** UNCLASSIFIED ***
                                                  *******************
                                                                                    PAGE 1
     EXTERNAL PROCEDURES REPORT

CALLER          EXT. PROCEDURE
-------------------------------------
AUDIT           ROUND
COPYTSK         INCK
FINDTASK        OUTOUT
FINDTASK        INCR
INIVAR          INCR
OUTFUN          OUTOUT
OUTFUN          CENTER
OUTPATH         CENTER
OUTTASK         OUTOUT
PATCH           DECR
PRUNE           DECR
REPORTD         REWIND
SORTTASKS       SWAP
STAMP           INIOUT
TASKDUMP        OUTOUT

                                                  *******************
                                                  *** UNCLASSIFIED ***
                                                  *******************
```

Figure 3-7. STAMP External Procedures Report

```
                                                       *******************
                                                       *** UNCLASSIFIED ***
                                                       *******************
--------------------------------------------------------------------------------------------------------------PAGE 11

                                        STRUCTURE CHART FOR STAMP
LEVEL OF CALL
0           1           2           3           4             5           6          7           8          9          10
----------------------------------------------------------------------------------------------------------------------------------
STAMP
            INIOUT(EXTERNAL)
            INIVAR
                        INCR(EXTERNAL)
            AUDIT
                        ROUND(EXTERNAL)
            READPATH
            SORTTASKS
                        SWAP(EXTERNAL)
            FINDTASK
                        OUTOUT(EXTERNAL)
                        INCR(EXTERNAL)
            OUTFUN
                        OUTOUT(EXTERNAL)
                        CENTER(EXTERNAL)
            STRUCTURE
            OUTTASK
                        OUTOUT(EXTERNAL)
                        TASKDUMP
                                    OUTOUT(EXTERNAL)
            REPORTD
                        REWIND(EXTERNAL)
                        SRCHTASK
            COPYTSK
                        INCR(EXTERNAL)
                                                       *******************
                                                       *** UNCLASSIFIED ***
                                                       *******************
```

Figure 3-8. STAMP Structure Chart

```
                                              *******************
                                              *** UNCLASSIFIED ***
                                              *******************
                                                                                  PAGE 16

                                              FUNCTION LIST
FUNCTION    DECRIPTION                                          VERSION INFORMATION
---------------------------------------------------------------------------------------------------------
ADDALL      ADD A CONSTANT TO EACH MEMBER OF A LIST             (PAGE: 39)
DUMPMAP     DUMP DEBUG MAP DATA                                 (PAGE: 47)
MAPPATH     MAP PATHS FOR NODE NUMBERING                        (PAGE: 43)
MINIMIZE    MINIMIZE TIMING INFORMATION                         (PAGE: 34)
OUTPATH     OUTPUT HIPO CHART INFORMATION FOR PATHS             (PAGE: 33)
PACKTAB     REMOVE NULL ENTRIES AND RENUMBER TABLES             (PAGE: 29)
PATCH       DELETE NODES AND RECONNECT PATHS                    (PAGE: 41)
PATHDUMP    OUTPUT HIPO CHART INFORMATION FOR ONE PATH          (PAGE: 29)
PRUNE       REDUCE THE DETAIL FOR MODULE FLOW                   (PAGE: 36)
STAMP       STRUCTURE AND TIMING ANALYSIS MODELING PROGRAM      (PAGE: 1)
SUBRCL      COLLECT ALL SUBROUTINE CALLS FOR A MODULE           (PAGE: 46)
TRACCALL    OUTPUT STRUCTURE CHART MODULE LISTS                 (PAGE: 26)
VARPROC     PROCESS VARIABLES FOR EACH MODULE                   (PAGE: 52)
XADD        EACH MEMBER OF ONE LIST TO EACH MEMBER OF OTHER     (PAGE: 40)
                                              *******************
                                              *** UNCLASSIFIED ***
                                              *******************
```

Figure 3-9. STAMP Function List

d. Module List (see figure 3-10). This report lists all subroutines and entry points in the program being examined. It provides the following information for each module:

(1) Module name

(2) Type of module: S = subroutine
T = task
E = entry point

(3) Page number refers to the page on which this module can be found in the listing provided by AUTOxxxx.

(4) Description of module is taken from the first line of the module.

(5) Call level shows both the minimal (top) and maximal (bottom) "distance" of the module from the main program as determined from the structure chart.

(6) Version information is taken directly from the master file.

e. HIPO Diagram (see figure 3-11). This chart shows, for each module, the following information: module name, a list of all modules which call this module; a description of the module; and a list of all program segments of the module, where a program segment is a contiguous group of records which contains no branches except possibly on the last record of the segment.

For each segment, the following are shown:

(1) Inputs: This is a list of all variables which are referenced within the segment.

(2) Processing: This is expressed by the string:

ll: label$_1$(S$_1$) label$_2$(S$_2$): <firstline-lastline> branchpoint ll is a sequence number.

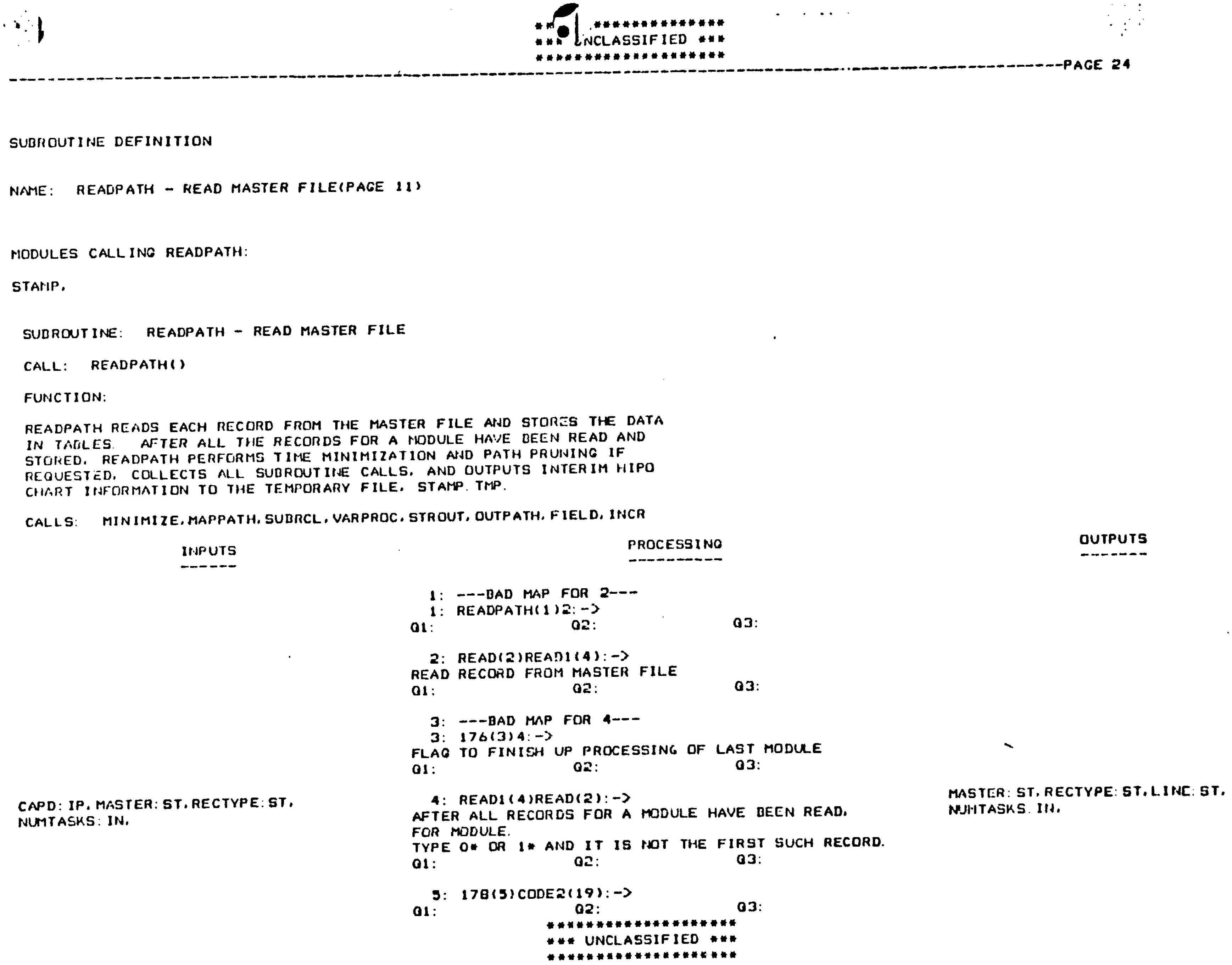
Label$_1$ is the line number of the first record of the segment.

S$_1$ is the segment number. (These are assigned sequentially in the order in which they occur in the module being scrutinized).

Label$_2$ is the line number of the segment to which program control is transferred after this segment of instructions has been executed. If program control can be transferred to more than one segment from this segment (such as in a conditional branch), then the information for this segment will be repeated once for each branch point. (That is, only ll, label$_2$, S$_2$, and branchpoint will change).

```
                                                          ***** *********
                                                          *** UNCLASSIFIED ***
                                                          *******************
----------------------------------------------------------------------------------------PAGE 17

                                                   MODULE LIST
             TASK/                                                          CALL LEVEL
MODULE NAME  SUBRTN PAGE  DECRIPTION                                        TOP  BOTTOM VERSION INFORMATION
-------------------------------------------------------------------------------------------------------------
ADDALL        T      39   ADD A CONSTANT TO EACH MEMBER OF A LIST            0    0
AUDIT         S      11   DUMP PROGRAM STATISTICS TO TERMINAL                1    1
COPYTSK       S      26   CREATE A NEW TASK TABLE TO SAVE SPACE              1    1
DUMPMAP       T      47   DUMP DEBUG MAP DATA                                0    0
FINDTASK      S      18   OUTPUT EXTERNAL PROCEDURES REPORT                  1    1
INIVAR        S      50   STORE VARIABLES TO INCLUDE OR ELIMINATE            1    1
MAPPATH       T      43   MAP PATHS FOR NODE NUMBERING                       0    0
MINIMIZE      T      34   MINIMIZE TIMING INFORMATION                        0    0
OUTFUN        S      20   OUTPUT FUNCTION LIST                               1    1
OUTPATH       T      33   OUTPUT HIPO CHART INFORMATION FOR PATHS            0    0
OUTTASK       S      19   OUTPUT MODULE LIST                                 1    1
PACKTAB       T      29   REMOVE NULL ENTRIES AND RENUMBER TABLES            0    0
PATCH         T      41   DELETE NODES AND RECONNECT PATHS                   0    0
PATHDUMP      T      29   OUTPUT HIPO CHART INFORMATION FOR ONE PATH         0    0
PRUNE         T      36   REDUCE THE DETAIL FOR MODULE FLOW                  0    0
READPATH      S      11   READ MASTER FILE                                   1    1
REPORTD       S      23   OUTPUT HIPO CHARTS                                 1    1
SORTTASKS     S      17   SORT TASK TABLE USING CACM ALGORITHM NO 201        1    1
SRCHTASK      S      49   BINARY SEARCH ON TABLE FOR GIVEN TASK NAME         1    2
STAMP         T      1    STRUCTURE AND TIMING ANALYSIS MODELING PROGRAM     0    0
STRUCTURE     S      22   OUTPUT STRUCTURE CHARTS                            1    1
SUBRCL        T      46   COLLECT ALL SUBROUTINE CALLS FOR A MODULE          0    0
TASKDUMP      S      21   OUTPUT MODULE LIST INFO FOR ONE MODULE             2    2
TRACCALL      T      26   OUTPUT STRUCTURE CHART MODULE LISTS                0    0
VARCLIP       S      51   ELIMINATE UNNECESSARY VARIABLES                    1    1
VARPROC       T      52   PROCESS VARIABLES FOR EACH MODULE                  0    0
XADD          T      40   EACH MEMBER OF ONE LIST TO EACH MEMBER OF OTHER    0    0
                                                          *******************
                                                          *** UNCLASSIFIED ***
                                                          *******************
```

Figure 3-10. STAMP Module List



```
                                                   *** ***************
                                                   *** UNCLASSIFIED ***
                                                   ********************
---------------------------------------------------------------------------------------PAGE 24


SUBROUTINE DEFINITION

NAME:  READPATH - READ MASTER FILE(PAGE 11)


MODULES CALLING READPATH:

STAMP,

 SUBROUTINE:  READPATH - READ MASTER FILE

 CALL:  READPATH()

 FUNCTION:

 READPATH READS EACH RECORD FROM THE MASTER FILE AND STORES THE DATA
 IN TABLES.  AFTER ALL THE RECORDS FOR A MODULE HAVE BEEN READ AND
 STORED, READPATH PERFORMS TIME MINIMIZATION AND PATH PRUNING IF
 REQUESTED, COLLECTS ALL SUBROUTINE CALLS, AND OUTPUTS INTERIM HIPO
 CHART INFORMATION TO THE TEMPORARY FILE, STAMP.TMP.

 CALLS:  MINIMIZE,MAPPATH,SUBRCL,VARPROC,STROUT,OUTPATH,FIELD,INCR
                   INPUTS                          PROCESSING                          OUTPUTS
                   ------                          ----------                          -------

                                   1: ---BAD MAP FOR 2---
                                   1: READPATH(1)2:->
                                 Q1:           Q2:           Q3:

                                   2: READ(2)READ1(4):->
                                 READ RECORD FROM MASTER FILE
                                 Q1:           Q2:           Q3:

                                   3: ---BAD MAP FOR 4---
                                   3: 176(3)4:->
                                 FLAQ TO FINISH UP PROCESSING OF LAST MODULE
                                 Q1:           Q2:           Q3:

 CAPD:IP,MASTER:ST,RECTYPE:ST,     4: READ1(4)READ(2):->                            MASTER:ST,RECTYPE:ST,LINE:ST,
 NUMTASKS:IN,                    AFTER ALL RECORDS FOR A MODULE HAVE BEEN READ,     NUMTASKS:IN,
                                 FOR MODULE.
                                 TYPE 0* OR 1* AND IT IS NOT THE FIRST SUCH RECORD.
                                 Q1:           Q2:           Q3:

                                   5: 178(5)CODE2(19):->
                                 Q1:           Q2:           Q3:
                                            ********************
                                            *** UNCLASSIFIED ***
                                            ********************
```

Figure 3-11. STAMP HIPO Chart

S_2 is the segment number to which control is transferred.

Firstline-lastline shows the first and last records of this segment.

Branchpoint is the record to which control is transferred.

The first line is followed by a second line with three values:
Q_1: This is the computer CPU time this segment consumes.
Q_2 is the number of assembly statements in the segment; and
Q_3 is the number of storage words in the segment.

(3) Outputs: This is a list of all variables which are modified by the segment.

3.3.3 SOFTPRO. SOFTPRO performs software quality analysis. Figure 3-12 shows the data flow between SOFTPRO and the rest of the PFA system.

SOFTPRO reads the master file and software standards file and generates two reports. The software profile reports for each module list the violations of software standards as defined by the user in the software standards file. The profile summary contains the list of standards, the number and percent of modules meeting each standard, the number and percent of modules violating each standard, and the reference to the regulation containing each standard.

3.3.3.1 SOFTPRO Input Requirements. The encoder program AUTOxxxx must have been run to create the master file. The software standards file must also have been created.

3.3.3.2 SOFTPRO Initiation Procedures. To run the SOFTPRO program, the user should enter, from an interactive terminal connected to the VAX, the command.

```
$ PFA
(respond to menu prompt for SOFTPRO)
```

SOFTPRO will then prompt the user for the following information:

COMMAND/RESPONSE	EXPLANATION
ENTER THE MASTER FILE NAME: M1.MST	This is the name of the master file that will be processed during this run.
ENTER THE STANDARDS FILE NAME: M1.STD	This is the name of the standards file to be referenced during this run.
DO YOU WANT THE FULL REPORT (Y/N)? Y	A full report includes a software standards violations details report for each module as well as the software standards violations summary report for the entire program.

SOFTPRO will then proceed to generate the software profile reports.

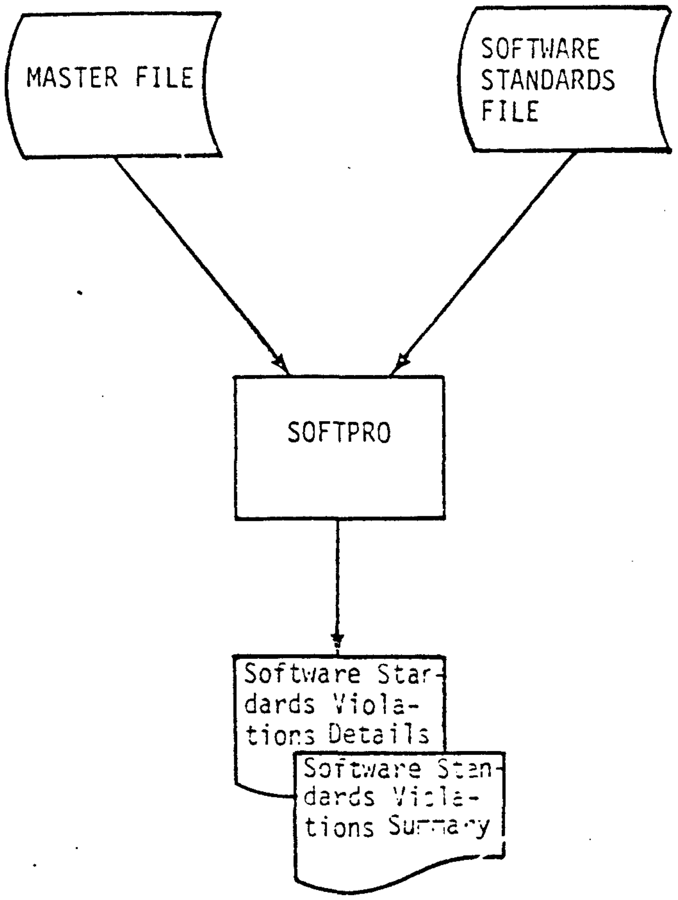

MASTER FILE
SOFTWARE STANDARDS FILE
SOFTPRO
Software Standards Violations Details
Software Standards Violations Summary

Figure 3-12. SOFTPRO Data Flow

3.3.3.3 Files Used in SOFTPRO Processing. The report output from SOFTPRO is placed in a file named PROFILE.REP.

3.3.3.4 SOFTPRO Recovery and Error Correction Procedures. SOFTPRO has the following error conditions:

a. Message: "STANDARD REFERENCE IN STANDARD LINE ____" IS NOT FOUND

This means that a standard reference in the calculation field of the indicated line of the standards file cannot be found.

b. Message: "IN STANDARDS FILE, ERROR IN SPECIFYING CALCULATION FOR STANDARD LINE ____"

This means that the calculation field of the indicated line of the standards file is in error.

c. Message: "IN THE STANDARD FILE, LINE ____ SPECIFIES A CALCULATION FOR A BASIC STANDARD"

This means that for the indicated line in the standards file, the standard number is between 1 and 14, indicating that it is a basic standard, but the calculation field contradicts that by providing a calculation.

d. Message: "IN THE STANDARDS FILE, LINE ____ SPECIFIES A STANDARD WHICH IS NOT BASIC BUT IT HAS NO CALUCLATION"

This means that for the indicated line in the standards file the standard number is not between 1 and 14, indicating that it is not a basic standard, but there was no calculation using basic standards included in the calculation field.

3.3.3.5 SOFTPRO Limitations.

a. A maximum of 14 basic standards can be entered on the standards file. A basic standard is one directly calculated by the encoder, AUTOxxxx. It must have an identifier of from 1 to 14 in the standards file.

b. Module names must be from 1 to 10 characters long.

c. The maximum number of standards allowed to be input is 30.

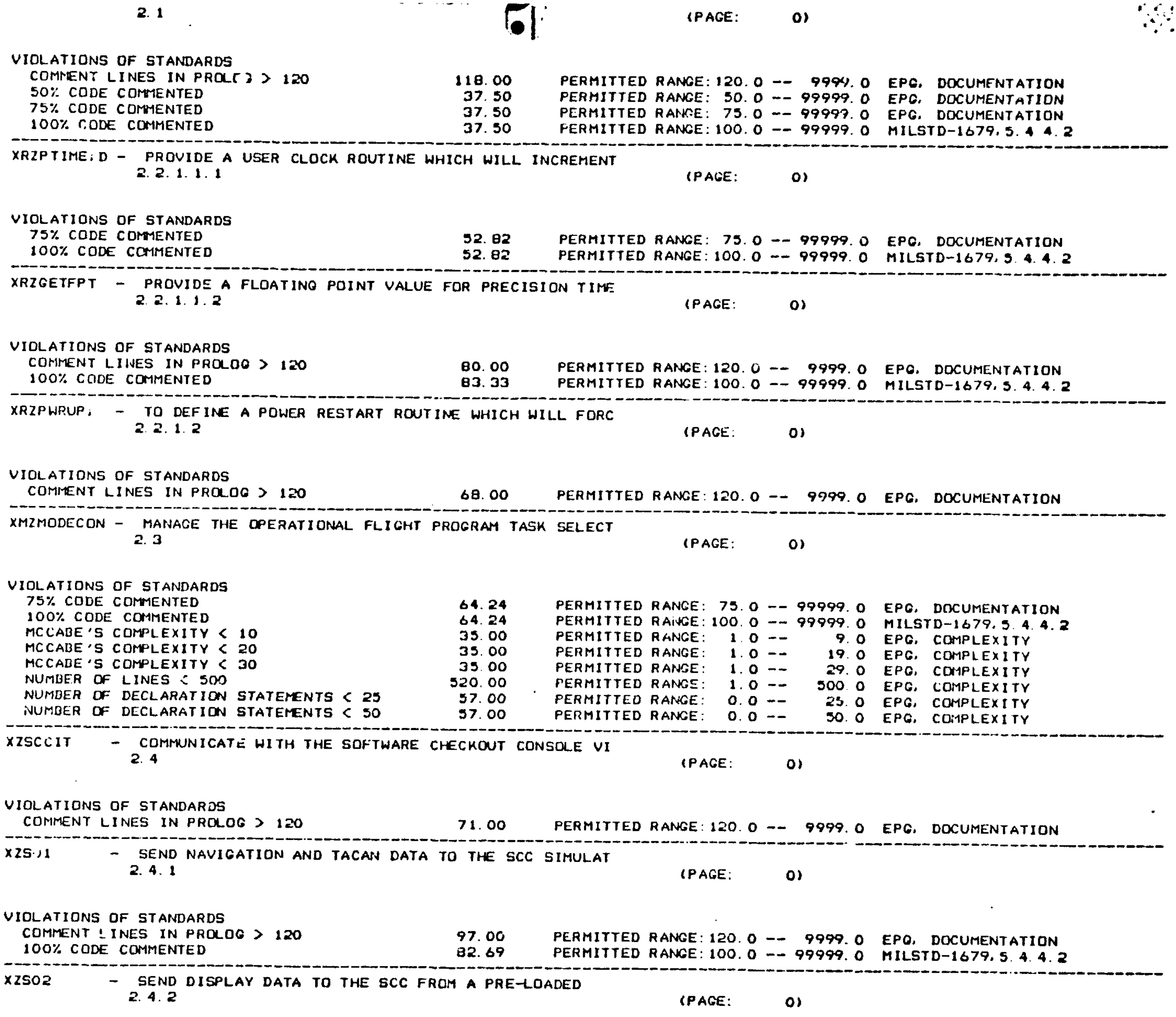
d. The maximum number of operators allowed in a standard expression is 20.

e. The maximum number of subroutines allowed in the master file is 200.

The last three limitations are program parameters which may be increased by modifying the parameters in the source code and recompiling and linking.

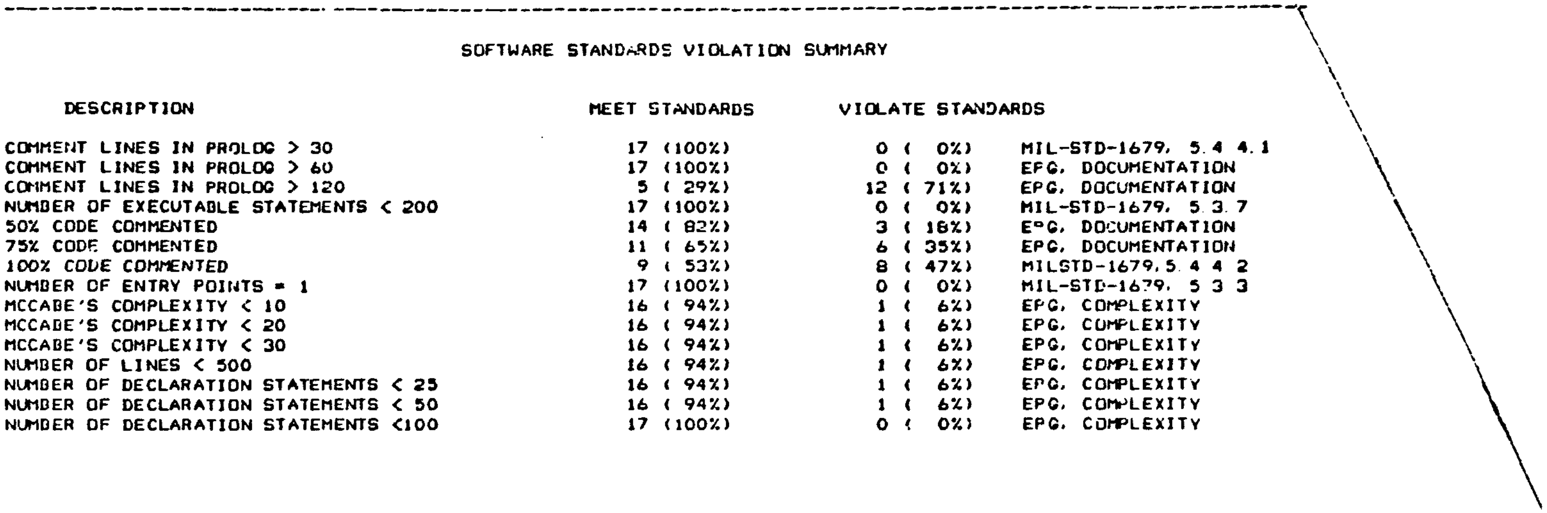
3.3.3.6 SOFTPRO Sample Outputs.

a. Software Standards Violations Details Report (see figure 3-13). This report lists each module of the file being analyzed. For each module



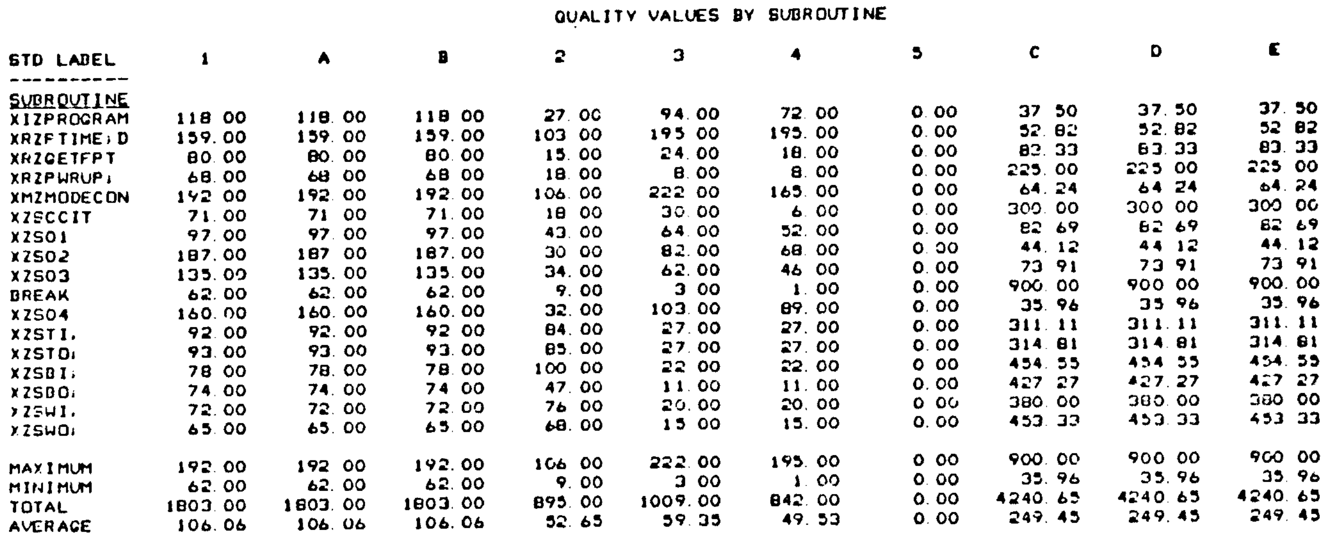
```
2.1                                                              (PAGE:      0)

VIOLATIONS OF STANDARDS
  COMMENT LINES IN PROLC] > 120              118.00    PERMITTED RANGE: 120.0 --  9999.0  EPG, DOCUMENTATION
  50% CODE COMMENTED                          37.50    PERMITTED RANGE:  50.0 -- 99999.0  EPG, DOCUMENTATION
  75% CODE COMMENTED                          37.50    PERMITTED RANGE:  75.0 -- 99999.0  EPG, DOCUMENTATION
  100% CODE COMMENTED                         37.50    PERMITTED RANGE: 100.0 -- 99999.0  MILSTD-1679,5.4.4.2
-----------------------------------------------------------------------------------------------------------
XRZPTIME;D -  PROVIDE A USER CLOCK ROUTINE WHICH WILL INCREMENT
              2.2.1.1.1                                          (PAGE:      0)

VIOLATIONS OF STANDARDS
  75% CODE COMMENTED                          52.82    PERMITTED RANGE:  75.0 -- 99999.0  EPG, DOCUMENTATION
  100% CODE COMMENTED                         52.82    PERMITTED RANGE: 100.0 -- 99999.0  MILSTD-1679,5.4.4.2
-----------------------------------------------------------------------------------------------------------
XRZGETFPT  -  PROVIDE A FLOATING POINT VALUE FOR PRECISION TIME
              2.2.1.1.2                                          (PAGE:      0)

VIOLATIONS OF STANDARDS
  COMMENT LINES IN PROLOG > 120               80.00    PERMITTED RANGE: 120.0 --  9999.0  EPG, DOCUMENTATION
  100% CODE COMMENTED                         83.33    PERMITTED RANGE: 100.0 -- 99999.0  MILSTD-1679,5.4.4.2
-----------------------------------------------------------------------------------------------------------
XRZPWRUP,  -  TO DEFINE A POWER RESTART ROUTINE WHICH WILL FORC
              2.2.1.2                                            (PAGE:      0)

VIOLATIONS OF STANDARDS
  COMMENT LINES IN PROLOG > 120               68.00    PERMITTED RANGE: 120.0 --  9999.0  EPG, DOCUMENTATION
-----------------------------------------------------------------------------------------------------------
XMZMODECON -  MANAGE THE OPERATIONAL FLIGHT PROGRAM TASK SELECT
              2.3                                                (PAGE:      0)

VIOLATIONS OF STANDARDS
  75% CODE COMMENTED                          64.24    PERMITTED RANGE:  75.0 -- 99999.0  EPG, DOCUMENTATION
  100% CODE COMMENTED                         64.24    PERMITTED RANGE: 100.0 -- 99999.0  MILSTD-1679,5.4.4.2
  MCCABE'S COMPLEXITY < 10                    35.00    PERMITTED RANGE:   1.0 --     9.0  EPG, COMPLEXITY
  MCCABE'S COMPLEXITY < 20                    35.00    PERMITTED RANGE:   1.0 --    19.0  EPG, COMPLEXITY
  MCCABE'S COMPLEXITY < 30                    35.00    PERMITTED RANGE:   1.0 --    29.0  EPG, COMPLEXITY
  NUMBER OF LINES < 500                      520.00    PERMITTED RANGE:   1.0 --   500.0  EPG, COMPLEXITY
  NUMBER OF DECLARATION STATEMENTS < 25       57.00    PERMITTED RANGE:   0.0 --    25.0  EPG, COMPLEXITY
  NUMBER OF DECLARATION STATEMENTS < 50       57.00    PERMITTED RANGE:   0.0 --    50.0  EPG, COMPLEXITY
-----------------------------------------------------------------------------------------------------------
XZSCCIT    -  COMMUNICATE WITH THE SOFTWARE CHECKOUT CONSOLE VI
              2.4                                                (PAGE:      0)

VIOLATIONS OF STANDARDS
  COMMENT LINES IN PROLOG > 120               71.00    PERMITTED RANGE: 120.0 --  9999.0  EPG, DOCUMENTATION
-----------------------------------------------------------------------------------------------------------
XZS01      -  SEND NAVIGATION AND TACAN DATA TO THE SCC SIMULAT
              2.4.1                                              (PAGE:      0)

VIOLATIONS OF STANDARDS
  COMMENT LINES IN PROLOG > 120               97.00    PERMITTED RANGE: 120.0 --  9999.0  EPG, DOCUMENTATION
  100% CODE COMMENTED                         82.69    PERMITTED RANGE: 100.0 -- 99999.0  MILSTD-1679,5.4.4.2
-----------------------------------------------------------------------------------------------------------
XZS02      -  SEND DISPLAY DATA TO THE SCC FROM A PRE-LOADED
              2.4.2                                              (PAGE:      0)
```

Figure 3-13. SOFTPRO Standards Violations Details Report



```
                          SOFTWARE STANDARDS VIOLATION SUMMARY

     DESCRIPTION                                 MEET STANDARDS      VIOLATE STANDARDS

COMMENT LINES IN PROLOG > 30                       17 (100%)          0 (  0%)    MIL-STD-1679, 5.4 4.1
COMMENT LINES IN PROLOG > 60                       17 (100%)          0 (  0%)    EPG, DOCUMENTATION
COMMENT LINES IN PROLOG > 120                       5 ( 29%)         12 ( 71%)    EPG, DOCUMENTATION
NUMBER OF EXECUTABLE STATEMENTS < 200              17 (100%)          0 (  0%)    MIL-STD-1679, 5.3.7
50% CODE COMMENTED                                 14 ( 82%)          3 ( 18%)    EPG, DOCUMENTATION
75% CODE COMMENTED                                 11 ( 65%)          6 ( 35%)    EPG, DOCUMENTATION
100% CODE COMMENTED                                 9 ( 53%)          8 ( 47%)    MILSTD-1679,5.4 4 2
NUMBER OF ENTRY POINTS = 1                         17 (100%)          0 (  0%)    MIL-STD-1679, 5 3 3
MCCABE'S COMPLEXITY < 10                           16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
MCCABE'S COMPLEXITY < 20                           16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
MCCABE'S COMPLEXITY < 30                           16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
NUMBER OF LINES < 500                              16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
NUMBER OF DECLARATION STATEMENTS < 25              16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
NUMBER OF DECLARATION STATEMENTS < 50              16 ( 94%)          1 (  6%)    EPG, COMPLEXITY
NUMBER OF DECLARATION STATEMENTS <100              17 (100%)          0 (  0%)    EPG, COMPLEXITY
```

Figure 3-14. SOFTPRO Standards Violations Summary Report



```
                                        QUALITY VALUES BY SUBROUTINE

STD LABEL        1          A          B          2          3          4          5          C          D          E
----------
SUBROUTINE
XIZPROGRAM    118.00     118.00     118.00      27.00      94.00      72.00      0.00      37.50      37.50      37.50
XRZFTIME;D    159.00     159.00     159.00     103.00     195.00     195.00      0.00      52.82      52.82      52.82
XRZGETFPT      80.00      80.00      80.00      15.00      24.00      18.00      0.00      83.33      83.33      83.33
XRZPWRUP;      68.00      68.00      68.00      18.00       8.00       8.00      0.00     225.00     225.00     225.00
XMZMODECON    192.00     192.00     192.00     106.00     222.00     165.00      0.00      64.24      64.24      64.24
XZSCCIT        71.00      71.00      71.00      18.00      30.00       6.00      0.00     300.00     300.00     300.00
XZSO1          97.00      97.00      97.00      43.00      64.00      52.00      0.00      82.69      82.69      82.69
XZSO2         187.00     187.00     187.00      30.00      82.00      68.00      0.00      44.12      44.12      44.12
XZSO3         135.00     135.00     135.00      34.00      62.00      46.00      0.00      73.91      73.91      73.91
BREAK          62.00      62.00      62.00       9.00       3.00       1.00      0.00     900.00     900.00     900.00
XZSO4         160.00     160.00     160.00      32.00     103.00      89.00      0.00      35.96      35.96      35.96
XZSTI.         92.00      92.00      92.00      84.00      27.00      27.00      0.00     311.11     311.11     311.11
XZSTO;         93.00      93.00      93.00      85.00      27.00      27.00      0.00     314.81     314.81     314.81
XZSBI;         78.00      78.00      78.00     100.00      22.00      22.00      0.00     454.55     454.55     454.55
XZSBO;         74.00      74.00      74.00      47.00      11.00      11.00      0.00     427.27     427.27     427.27
XZSWI.         72.00      72.00      72.00      76.00      20.00      20.00      0.00     380.00     380.00     380.00
XZSWO;         65.00      65.00      65.00      68.00      15.00      15.00      0.00     453.33     453.33     453.33

MAXIMUM       192.00     192.00     192.00     106.00     222.00     195.00      0.00     900.00     900.00     900.00
MINIMUM        62.00      62.00      62.00       9.00       3.00       1.00      0.00      35.96      35.96      35.96
TOTAL        1803.00    1803.00    1803.00     895.00    1009.00     842.00      0.00    4240.65    4240.65    4240.65
AVERAGE       106.06     106.06     106.06      52.65      59.35      49.53      0.00     249.45     249.45     249.45
```

Figure 3-15. SOFTPRO Quality Values Report

```
                                   S T A N D A R D S

LABEL   DESCRIPTION                                  PERMITTED RANGE       STANDARD REFERENCE

  1     COMMENT LINES IN PROLOG > 30                  30.0 --   9999 0     MIL-STD-1679.  5 4.4.1
  A     COMMENT LINES IN PROLOG > 60                  60.0 --   9999 0     EPG.  DOCUMENTATION
  B     COMMENT LINES IN PROLOG > 120                120 0 --   9999 0     EPG.  DOCUMENTATION
  2     NUMBER OF NON-BLANK COMMENT LINES              0 0 --   9999 0     EPG.  DOCUMENTATION
  3     NUMBER OF NON-COMMENT LINES                    0 0 --   9999 0
  4     NUMBEP OF EXECUTABLE STATEMENTS < 200          0 0 --    200 0     MIL-STD-1679.  5 3 7
  5     NUMBER OF STATEMENTS WITH EMBEDDED COMME       0.0 --   9999.0     EPG.  DOCUMENTATION
  C     50% CODE COMMENTED                            50 0 --  99999 0     EPG.  DOCUMENTATION
  D     75% CODE COMMENTED                            75 0 --  99999 0     EPG.  DOCUMENTATION
  E     100% CODE COMMENTED                          100.0 --  99999 0     MILSTD-1679.5 4 4.2
  6     NUMBE' OF ENTRY POINTS = 1                     0 0 --      1 0     MIL-STD-1679.  5 3.3
  7     HALLSTEADS COMPLEXITY < 5                      1 0 --      5 0     EPG.  COMPLEXITY
  8     KNOTS COMPLEXITY < 3                           0 0 --      3 0     EPG.  STRUCTURE
  9     MCCABE'S COMPLEXITY < 10                       1 0 --      9 0     EPG.  COMPLEXITY
  F     MCCABE'S COMPLEXITY < 20                       1 0 --     19 0     EPG.  COMPLEXITY
  G     MCCABE'S COMPLEXITY < 30                       1.0 --     29 0     EPG.  COMPLEXITY
  H     NUMBER OF LINES < 500                          1 0 --    500 0     EPG.  COMPLEXITY
  I     NUMBER OF DECLARATION STATEMENTS < 25          0 0 --     25 0     EPG.  COMPLEXITY
  J     NUMBER OF DECLARATION STATEMENTS < 50          0 0 --     50 0     EPG.  COMPLEXITY
  K     NUMBER OF DECLARATION STATEMENTS <100          0 0 --    100 0     EPG.  COMPLEXITY
 10     FUTURE DEVELOPMENT (TOP DOWN STRUCTURE)        1.0 --      3 0     EPG.  STRUCTURE
 11     FUTURE DEV.                                    1.0 --    100 0     EPG.  NEW METRIC
```

Figure 3-15. SOFTPRO Quality Values Report (Cont.)

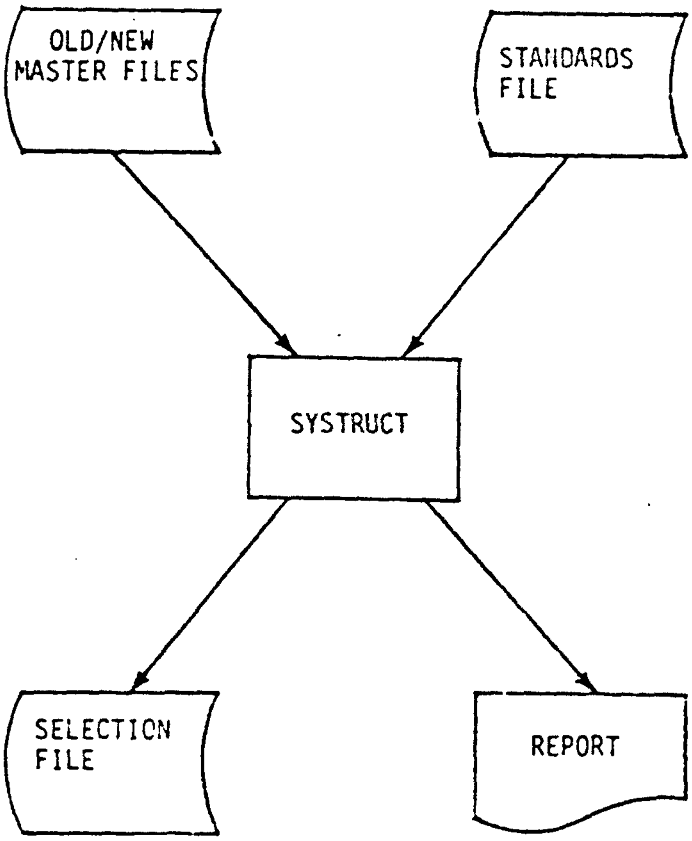


Figure 3-16. SYSTRUCT Data Flow

all standards defined in the software standards file which have been violated by that module are listed. The value of the standard calculated for that module is shown, along with the permitted range of the standard.

b. Software Standards Violations Summary Report (see figure 3-14). For each standard defined in the software standards file, the number and percentage of modules which met that standard are shown, and the number and percentage of modules which violated that standard are shown. Additionally, the quality values are summarized for all the modules and listed individually for each module (see figure 3-15).

3.3.4 SYSTRUCT. SYSTRUCT provides analysis of system configuration changes. Figure 3-16 shows the data flow between SYSTRUCT and the rest of the PFA system.

SYSTRUCT reads two master files representing different versions of the software system under analysis. The SYSTRUCT report lists changes in variables used by each module, changes in software quality parameters (the ones reported by SOFTPRO), and modules which have been added to or deleted from the current version of the software being analyzed. The changes are of two types: those affecting executable code and those affecting code documentation. When changes affect executable code, a selection file containing a list of module changes is generated for use by PATCHANA.

3.3.4.1 SYSTRUCT Input Requirements. SYSTRUCT requires two master files as input, one for each version of the program to be analyzed. See paragraph 3.2.3 and appendix B for the master file description.

SYSTRUCT also requires a standards file for the standards file description. A description of the file and its layout is given in paragraph 3.2.3 and in appendix B.

3.3.4.2 SYSTRUCT Initiation Procedures. Following is a description of a SYSTRUCT terminal session.

SYSTRUCT may be run from any interactive terminal connected to the VAX. Assuming that the user has logged onto the VAX with the correct password, he can run SYSTRUCT by entering the following command:

```
$PFA
(respond to menu prompt for SYSTRUCT)
```

SYSTRUCT will then prompt the user for the following information:

```
    COMMAND/RESPONSE                    EXPLANATION

ENTER OLD MASTER FILE NAME, WITH       The entered old and new master file names
EXTENSION                              are the names of the master files for
MOLD.MST                               the old and new versions of the programs
ENTER NEW MASTER FILE NAME, WITH       to be analyzed.
EXTENSION
MNEW.MST
```

COMMAND/RESPONSE	EXPLANATION
ENTER REPORT FILE NAME M.REP	The entered report file name is the name of the disk file to contain the output SYSTRUCT report.
ENTER STANDARDS FILE NAME M.STD	The entered standards file name is the name of the standards file to be referenced during this run.
ENTER SELECTION FILE NAME: M.SEL	The entered selection file name is the name of the disk file to contain the output list of modules having probable code changes.
DO YOU WANT STRUCTURE CHART? YES	If requested, a structure chart for the new version of a module will be output in the report on each module.

SYSTRUCT will then complete the report generation with no further prompts.

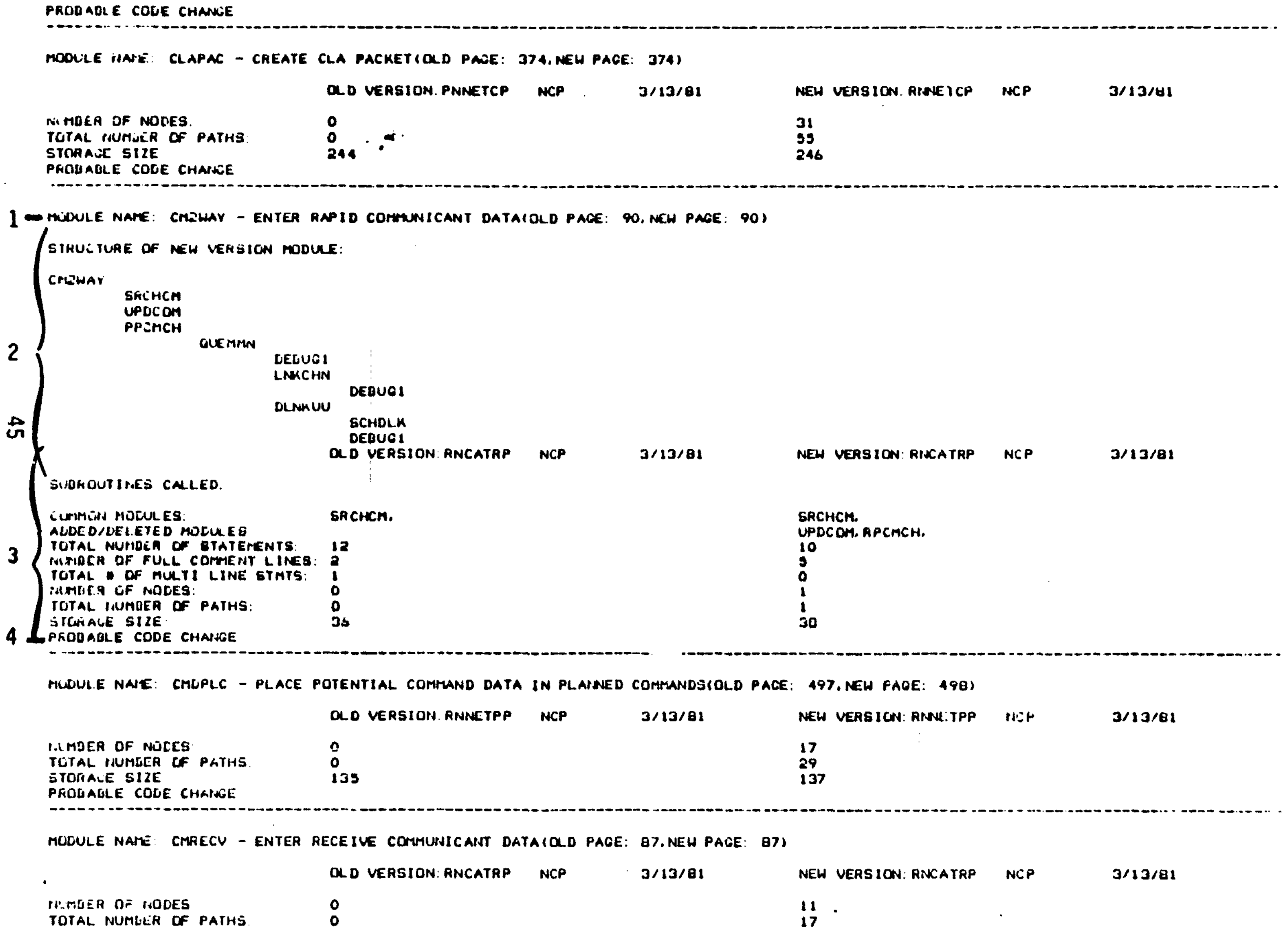
3.3.4.3 Files Used in SYSTRUCT Processing. No temporary files are used.

3.3.4.4 SYSTRUCT Recovery and Error Correction Procedures. In the event of a hardware error which terminates the program, the user must start the program from the beginning. In the event of a fatal software error while running SYSTRUCT, the user should check to make sure all files exist and are not attached to another program, and rerun SYSTRUCT.

3.3.4.5 SYSTRUCT Limitations. No limitations are known.

3.3.4.6 SYSTRUCT Sample Outputs. Figure 3-17 presents a sample portion of a SYSTRUCT report. This report shows, at module level, differences in subroutine calls, number of paths containing subroutine calls, and software quality metrics between two versions of a system. The report consists of a series of mini-reports, one for each module which has changed between versions. Each mini-report has two to four parts. First, the report begins with a header line containing the module name, description, and starting page numbers of both versions in the encoder listings. Second, the report contains an optional structure chart, similar to the STAMP structure chart, for the new version of the module. Third, the report contains a list of the differences between the versions of the module. The list is headed by the version information for the old and new modules. The list contains common, added, and deleted subroutines if changes in subroutine calls have been made, and software metrics with values that differ between versions. Fourth, the optional last line in the report describes the probable level of change in the module. If a software code metric value has changed, a code level change is indicated; if a software documentation metric value has changed, a documentation level change is indicated.

3.3.5 PATCHANA. PATCHANA performs software patch (source code change) analysis. Figure 3-18 shows the data flow between PATCHANA and the rest of the PFA system.



```
PRODABLE CODE CHANGE
----------------------------------------------------------------------------------------------------------------

MODULE NAME:  CLAPAC - CREATE CLA PACKET(OLD PAGE: 374,NEW PAGE: 374)

                                 OLD VERSION.PNNETCP   NCP         3/13/81          NEW VERSION.RNNE1CP   NCP         3/13/81

NUMBER OF NODES.                 0                                                  31
TOTAL NUMBER OF PATHS:           0                                                  55
STORAGE SIZE                     244                                                246
PROBABLE CODE CHANGE
----------------------------------------------------------------------------------------------------------------

MODULE NAME:  CM2WAY - ENTER RAPID COMMUNICANT DATA(OLD PAGE: 90,NEW PAGE: 90)
STRUCTURE OF NEW VERSION MODULE:

CM2WAY
      SRCHCM
      UPDCOM
      PPCMCH
            QUEMMN
                  DEBUG1
                  LNKCHN
                        DEBUG1
                  DLNKUU
                        SCHDLK
                        DEBUG1
                                 OLD VERSION:RNCATRP   NCP         3/13/81          NEW VERSION:RNCATRP   NCP         3/13/81

SUBROUTINES CALLED.

COMMON MODULES:                  SRCHCM,                                            SRCHCM,
ADDED/DELETED MODULES                                                               UPDCOM,RPCMCH,
TOTAL NUMBER OF STATEMENTS:      12                                                 10
NUMBER OF FULL COMMENT LINES:    2                                                  5
TOTAL # OF MULTI LINE STMTS:     1                                                  0
NUMBER OF NODES:                 0                                                  1
TOTAL NUMBER OF PATHS:           0                                                  1
STORAGE SIZE                     36                                                 30
PROBABLE CODE CHANGE
----------------------------------------------------------------------------------------------------------------

MODULE NAME:  CMDPLC - PLACE POTENTIAL COMMAND DATA IN PLANNED COMMANDS(OLD PAGE: 497,NEW PAGE: 498)

                                 OLD VERSION.RNNETPP   NCP         3/13/81          NEW VERSION:RNNETPP   NCP         3/13/81

NUMBER OF NODES                  0                                                  17
TOTAL NUMBER OF PATHS.           0                                                  29
STORAGE SIZE                     135                                                137
PROBABLE CODE CHANGE
----------------------------------------------------------------------------------------------------------------

MODULE NAME:  CMRECV - ENTER RECEIVE COMMUNICANT DATA(OLD PAGE: 87,NEW PAGE: 87)

                                 OLD VERSION:RNCATRP   NCP         3/13/81          NEW VERSION:RNCATRP   NCP         3/13/81

NUMBER OF NODES                  0                                                  11
TOTAL NUMBER OF PATHS.           0                                                  17
```

Figure 3-17. SYSTRUCT Report

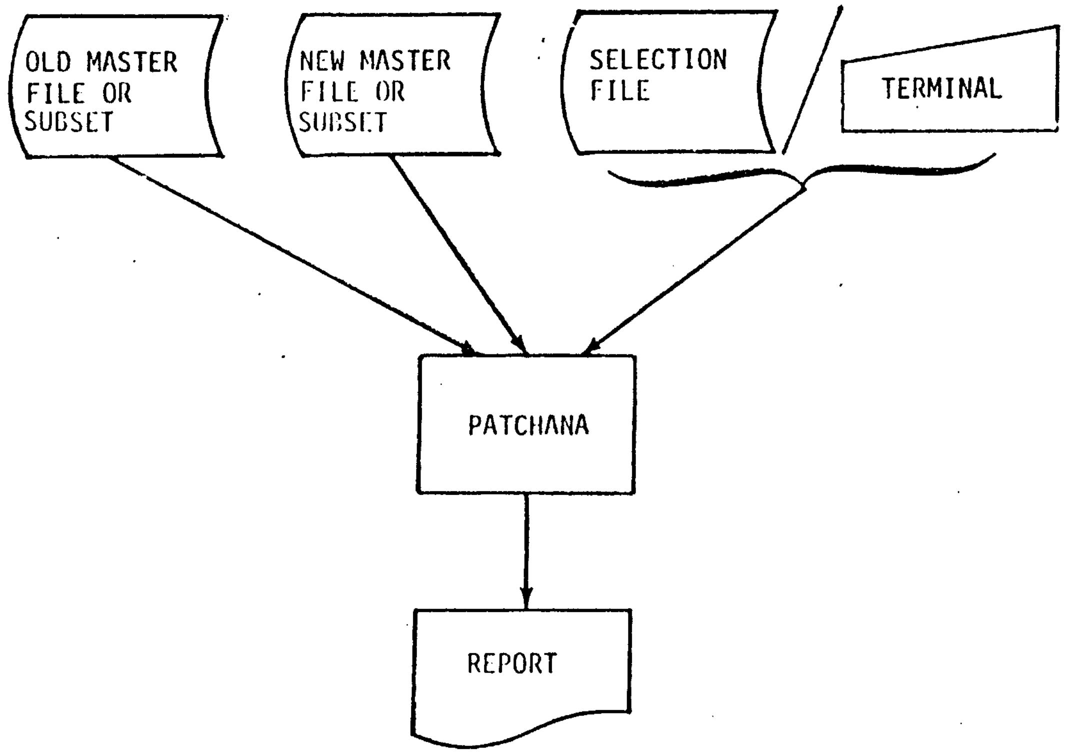


Figure 3-18. PATCHANA Data Flow

PATCHANA reads two master files or subsets of master files representing two versions of the software being analyzed. Selection can be made interactively or via the selection file, which contains a list of modules to be compared. The report lists the path segment structure for both versions side by side and indicates (with codes and arrows) the location of differences and what was changed for each version. When structural differences occur, the differences are flagged from their start either through the remainder of the module or to the point where the structures of the two versions are congruent. PATCHANA finds differences in structure, subroutine calls, variable usage, and the three quantifiers.

3.3.5.1 PATCHANA Input Requirements. PATCHANA requires as input two master files, or subsets of two master files, one for each version of the system to be analyzed. See paragraph 3.2.3 and appendix B for the master file description. If a subset of a master file is used, it must contain the data for all the modules to be analyzed.

PATCHANA also requires as input a list of modules to be analyzed. This list may be entered interactively, as described below, or the list may be input as a selection file, as described in paragraph 3.3.3.

3.3.5.2 PATCHANA Initiation Procedures. PATCHANA may be run from any interactive terminal connected to the VAX. Assuming that the user has logged onto the VAX with the correct password, he can run PATCHANA by entering the following command:

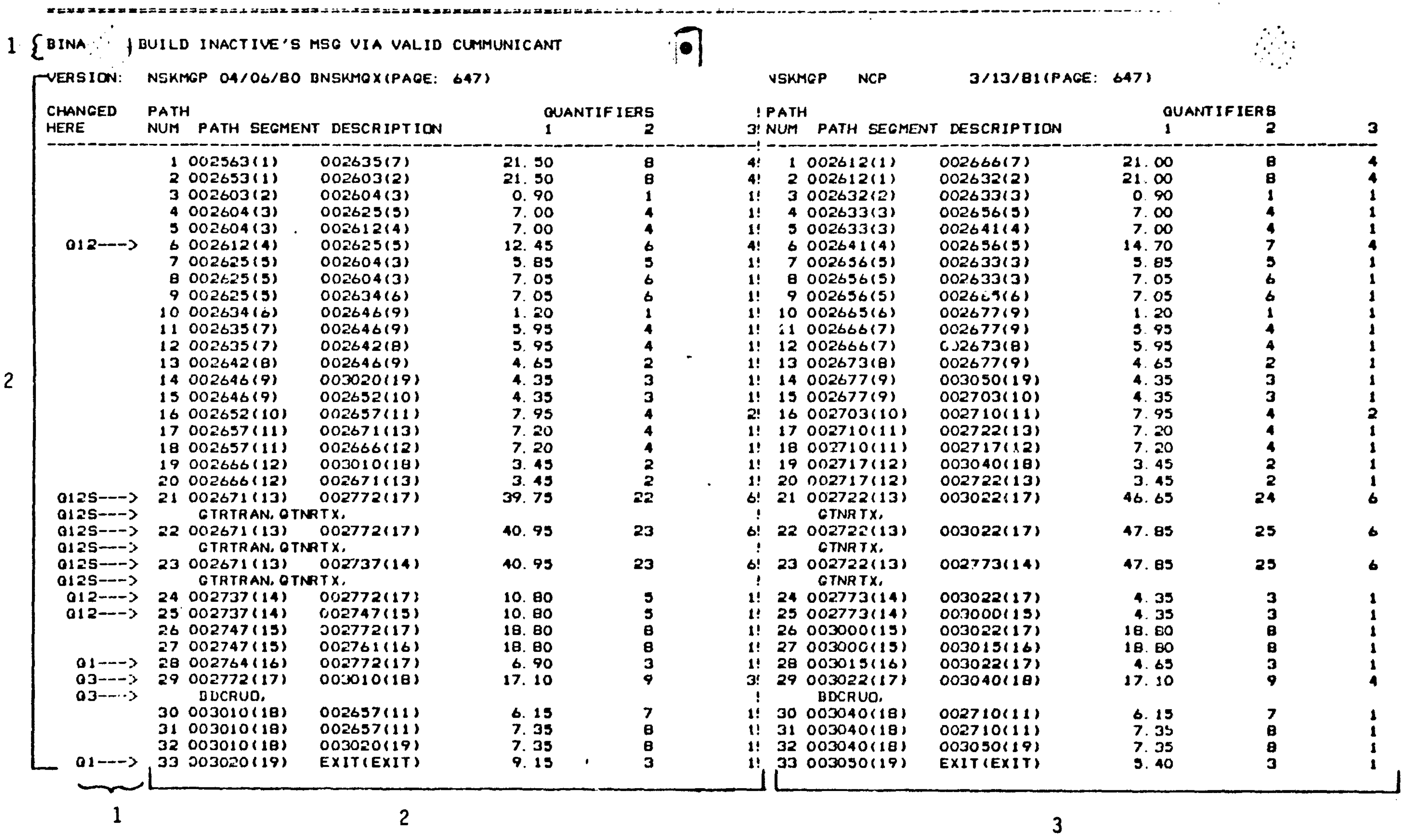
```
$ PFA
(respond to menu prompt for PATCHANA)
```

PATCHANA will then prompt the user for the following information:

PROMPT/RESPONSE	EXPLANATION
ENTER OLD MASTER FILE NAME: MOLD.MST	This is the name of the subset of the master file containing information on the old versions of the modules to be processed.
ENTER NEW MASTER FILE NAME: NEW.MST	New second master file to compare against old first.
ENTER REPORT FILE NAME: MST.LST	Report file where the results of the change analysis are stored.

3.3.5.3 PATCHANA Limitations. PATCHANA cannot process multiple entry points.

3.3.5.4 PATCHANA Sample Outputs. Figure 3-19 is a sample PATCHANA report. This report shows path level differences between module versions. The report consists of a series of mini-reports, one for each module. Each mini-report consists of two parts, a header, and a patch comparison list. The header contains the module name and description. The list consists of three sets of columns: a description of the old paths, a description of the new paths, and a description of the differences between the two paths.



```
1 {BINA    }BUILD INACTIVE'S MSG VIA VALID CUMMUNICANT
 VERSION:   NSKMGP 04/06/80 BNSKMQX(PAGE:  647)                                  NSKMGP    NCP        3/13/81(PAGE:  647)
 CHANGED    PATH                                          QUANTIFIERS           !PATH                                           QUANTIFIERS
 HERE       NUM  PATH SEGMENT  DESCRIPTION                 1           2       3!NUM  PATH SEGMENT  DESCRIPTION              1             2           3
 ---------------------------------------------------------------------------------!--------------------------------------------------------------------------
               1 002563(1)       002635(7)             21.50           8       4!  1 002612(1)       002666(7)            21.00           8           4
               2 002653(1)       002603(2)             21.50           8       4!  2 002612(1)       002632(2)            21.00           8           4
               3 002603(2)       002604(3)              0.90           1       1!  3 002632(2)       002633(3)             0.90           1           1
               4 002604(3)       002625(5)              7.00           4       1!  4 002633(3)       002656(5)             7.00           4           1
               5 002604(3)       002612(4)              7.00           4       1!  5 002633(3)       002641(4)             7.00           4           1
   Q12--->     6 002612(4)       002625(5)             12.45           6       4!  6 002641(4)       002656(5)            14.70           7           4
               7 002625(5)       002604(3)              5.85           5       1!  7 002656(5)       002633(3)             5.85           5           1
               8 002625(5)       002604(3)              7.05           6       1!  8 002656(5)       002633(3)             7.05           6           1
               9 002625(5)       002634(6)              7.05           6       1!  9 002656(5)       002665(6)             7.05           6           1
              10 002634(6)       002646(9)              1.20           1       1! 10 002665(6)       002677(9)             1.20           1           1
              11 002635(7)       002646(9)              5.95           4       1! 11 002666(7)       002677(9)             5.95           4           1
              12 002635(7)       002642(8)              5.95           4       1! 12 002666(7)       002673(8)             5.95           4           1
              13 002642(8)       002646(9)              4.65           2       1! 13 002673(8)       002677(9)             4.65           2           1
              14 002646(9)       003020(19)             4.35           3       1! 14 002677(9)       003050(19)            4.35           3           1
              15 002646(9)       002652(10)             4.35           3       1! 15 002677(9)       002703(10)            4.35           3           1
              16 002652(10)      002657(11)             7.95           4       2! 16 002703(10)      002710(11)            7.95           4           2
              17 002657(11)      002671(13)             7.20           4       1! 17 002710(11)      002722(13)            7.20           4           1
              18 002657(11)      002666(12)             7.20           4       1! 18 002710(11)      002717(12)            7.20           4           1
              19 002666(12)      003010(18)             3.45           2       1! 19 002717(12)      003040(18)            3.45           2           1
              20 002666(12)      002671(13)             3.45           2       1! 20 002717(12)      002722(13)            3.45           2           1
  Q12S--->    21 002671(13)      002772(17)            39.75          22       6! 21 002722(13)      003022(17)           46.65          24           6
  Q12S--->          GTRTRAN,QTNRTX,                                             !       GTNRTX,
  Q12S--->    22 002671(13)      002772(17)            40.95          23       6! 22 002722(13)      003022(17)           47.85          25           6
  Q12S--->          GTRTRAN,QTNRTX,                                             !       GTNRTX,
  Q12S--->    23 002671(13)      002737(14)            40.95          23       6! 23 002722(13)      002773(14)           47.85          25           6
  Q12S--->          GTRTRAN,QTNRTX,                                             !       GTNRTX,
   Q12--->    24 002737(14)      002772(17)            10.80           5       1! 24 002773(14)      003022(17)            4.35           3           1
   Q12--->    25 002737(14)      002747(15)            10.80           5       1! 25 002773(14)      003000(15)            4.35           3           1
              26 002747(15)      002772(17)            18.80           8       1! 26 003000(15)      003022(17)           18.80           8           1
              27 002747(15)      002761(16)            18.80           8       1! 27 003000(15)      003015(16)           18.80           8           1
    Q1--->    28 002764(16)      002772(17)             6.90           3       1! 28 003015(16)      003022(17)            4.65           3           1
    Q3--->    29 002772(17)      003010(18)            17.10           9       3! 29 003022(17)      003040(18)           17.10           9           4
    Q3--->          BDCRUD,                                                     !       BDCRUD,
              30 003010(18)      002657(11)             6.15           7       1! 30 003040(18)      002710(11)            6.15           7           1
              31 003010(18)      002657(11)             7.35           8       1! 31 003040(18)      002710(11)            7.35           8           1
              32 003010(18)      003020(19)             7.35           8       1! 32 003040(18)      003050(19)            7.35           8           1
    Q1--->    33 003020(19)      EXIT(EXIT)             9.15           3       1! 33 003050(19)      EXIT(EXIT)            5.40           3           1
```

Figure 3-19. PATCHANA Report

The description of the differences between the paths is first. The description consists of a series of flags indicating the types of differences found. The flags are:

Q[1][2][3]	Differences in quantifier(s) 1 and/or 2 and/or 3
P	Difference in successor paths
N	Difference in successor nodes
S	Difference in subroutine calls
V	Difference in variable lists

If no differences are found, this column will be blank. This description of the differences between paths will be repeated for every line of the path descriptions.

The description of the old paths is second, and the description of the new paths is third. The descriptions contain the same information and have the following format: Each description is headed by the version information for the module being described. This header is followed by six subcolumns. The first column contains the number of the path being described. The second column contains the label and, in parentheses, the number of the from node for the path. The third column contains the label and, in parentheses, the number of the to node for the path. If requested, the second and third columns may contain source sequence numbers rather than labels. Columns four through six contain the values of quantifiers 1, 2, and 3, respectively. Following the column information for each path, the path description contains a list of the subroutines called in the path and a list of tne variables used in the path. Each variable entry in the variable list consists of the variable name followed by a colon, optionally followed by variable type, followed by a colon, followed by variable use. The variable use entry must be one of the following:

D	Defined
R	Referenced
T	Tested
M	Modified

3.4 Utilization of System Outputs.

3.4.1 STAMP Reports.

a. The external procedures report (figure 3-7) lists subroutines which are called but are not part of the software being analyzed. These subroutines are usually a part of libraries supplied with the operating system or compiler for the system being analyzed. If subroutines are in the list, and should be in the software being analyzed, then those subroutines are missing.

b. The structure chart (figure 3-8) shows the subroutine call hierarchy, starting from task or program level (level 0) through each level of subroutine call to the lowest level. The subroutines called by the task level programs are level one, the subroutines they call are level two, etc. Each level is shown on the chart indented to the right of the higher level. The subroutine name is given and, as an option, a brief description is given. External subroutines and recursive subroutines are marked as such.

c. The function list (figure 3-9) lists the highest level procedures in the software under analysis, i.e., at the program or task level, procedures which are not called as subroutines.

d. The module list (figure 3-10) lists all procedures and entry points in alphabetical order. The columns in the report are procedure name, task/subroutine flag, page, description, top call level, bottom call level, and version information.

The procedure name contains the name of the task, subroutine, or entry point. The task/subroutine flag shows an "S" if the procedure is a subroutine, a "T" if the procedure is a task or is not called by another procedure, or an "E" if the procedure is an unused or uncalled entry point in a module with multiple entry points. The page and description are supplied by the master file, as determined by the encoder. The top call level is the highest level on which the procedure is called. This is zero for a task or unused entry point, one for procedures called by them, etc. The bottom level is the lowest level at which a subroutine is called. Again, tasks and unused entry points have a level of zero. Procedures which have little or no difference between top call level and bottom call level are most likely a "single use" or "limited use" routine. If there is much difference between the top call level and the bottom call level, the procedure is more likely a utility routine. The version information is taken directly from the master file.

e. The HIPO chart (figure 3-11) gives detailed information about each procedure. "Subroutine Definition" or "Task Definition" heads each procedure listing. This is followed by the procedure name, short description, page, and version information, all of which are extracted from the master file. These are followed by a list of modules calling the procedure; this is not shown if the procedure is a task. Next is a list of modules called by the procedure; this is only shown if the procedure makes subroutine calls. Next is the list of entry points (not shown on example). This list is produced only if the procedure contains multiple entry points. The prologue copied from the master file follows. Then, the inputs column lists the variables which are referenced (but not modified) by the procedure. Variable type information is included. Next, the processing column has a description of each path, using both labels produced by the encoder, and sequence numbers from the original source as processed by the encoder. This allows the user to see how the structure represented in the processing column relates to the original source code. The processing column also contains comments extracted from the original source by the encoder; quantifiers, which are quantities such as time, statement counts, etc.; and a list of subroutines called on each path. Finally, the outputs column lists all of the variables, with their variable type information, which are modified on each path. The inputs, processing, and outputs columns can be used to check the HIPO charts in the Program Design Document CPCI's for accuracy.

3.4.2 SOFTPRO Reports.

a. The SOFTPRO detailed report (figure 3-13) lists each procedure and the standards it violates. First, the name of the procedure, description, version information, and page number are extracted from the master file and printed. Then the software quality metrics for the procedure are compared against the software standards and the violations are listed. The first column of the list of violations contains the description of the standard. The next column contains the value provided (or calculated) for the software being analyzed. The next column has the "permitted range," which is the range of values which the value provided for the software must fall within to meet the standard. Finally, there is a reference to the document containing the standard (where applicable).

b. The SOFTPRO summary report (figure 3-14) summarizes the violations of the standards. The first column contains the description of the standard. The next column contains the number and percentage of procedures which met each standard. The next column contains the number and percent of procedures which violated each standard. The last column provides a reference to the document containing the standard. Additionally, the quality values are summarized for all the modules and listed individually for each module (see figure 3-15).

3.4.3 SYSTRUCT Report. The SYSTRUCT report (figure 3-17) shows differences in subroutine calls, variable usage, and software quality metrics between two versions of the software under analysis. First, the name, description, and page number of old and new versions are printed. If the option for new structure is set, a structure chart similar to the STAMP structure chart is printed for the structure of the new version of the procedure. Next a header is printed; this contains the version information for the old and new versions. If changes in subroutine calls are made, the subroutines called which are common to both versions are listed; then the subroutine calls which have been added or deleted are reported. Next, the software metrics for which values have changed are reported. There is also a flag with each metric to associate it with either a change in executable code or a change in program documentation. A change in executable code supersedes a change in program documentation. The probable type of change or "no change" is flagged upon completion of each procedure.

3.4.4 PATCHANA Report. The PATCHANA report (figure 3-19) lists the structure of two versions of a procedure side by side and shows where differences are. First the name and description of the procedure are given. Then a header containing the version information for the old and new versions of the procedure is printed. The rest of the report is in three sections: the change indicator, the old version, and the new version. The change indicator can flag either a structural change or a change in attributes for the particular path. A structural change is flagged by a "P" for path or an "N" for node, where the change is a change in the execution path due to branches being added or deleted. A change in attributes for a particular path is a change in the subroutines called, variables used, or one or more of the quantifers changed. These changes are flagged by "S", "V", or "Q" followed by the quantifier numbers. The old and new version sections have the same format--the path number, and path segment description followed by the values of the three quantifiers. If subroutines are called, they are listed on the next line.

APPENDIX A

TERMS AND ABBREVIATIONS

TERMS AND ABBREVIATIONS

COMPRO	Comment Processor.
DEC	Digital Equipment Corporation.
DECUS	DEC User Society.
Encoder	A program which reads the source code for the system being analyzed and creates a master file for use by other PFA programs.
External procedures	Procedures which exist outside of the software being analyzed, e.g., operating system utilities.
HIPO	Hierarchy plus Inputs, Processing, and Outputs.
HIPO chart	A chart relating inputs and outputs to the processing algorithm which uses or creates them.
MACROSPITBOL	A SNOBOL4 compatible interpreter for the VAX.
Master file	The file which contains a representation of the attributes of the software being analyzed.
Module	A separately compilable procedure or subroutine.
Node	A point of decision in execution path selection.
PATCHANA	Patch Analysis Program.
Path	A segment of instructions which does not contain any branches.
PFA	Program Flow Analyzer.
PFALIB	PFA Library.
Program structure	Graph of the sequence of all possible paths which may be executed within a program.
Quantifiers	A value which quantifies or totals some attribute on a path.
RATFOR/RATFIV	A structured FORTRAN translator.
SOFTPRO	Software Profile Program.
Software patch	A change in source code.
Software system	Consists of one or more computer programs which perform one or more related functions.

TERMS AND ABBREBIATIONS (Cont'd.)

STAMP	Structure, Timing, Analysis, Modeling Program.
Standards file	File which contains user-entered standards in a prescribed format.
Structure chart	Shows a subroutine call hierarchy in the form of a tree graph.
SYSTRUCT	System Structure Comparison Program.
TECOM	Test and Evaluation Command.
USAEPG	U.S. Army Electronic Proving Ground.

APPENDIX B

MASTER FILE

AND

STANDAROS FILE DESCRIPTIONS

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 0 *

	RECORD TYPE
	MODULE NAME
5	
10	
	MONTH
15	DAY
	YEAR
20	HOUR
	MINUTE
	MODULE DESCRIPTION
25	
30	
35	
40	
45	
50	

50	
	MODULE DESCRIPTION, CONT.
55	
60	
65	
70	

Figure B-1. Library Routine Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 1 *

	RECORD TYPE
	MODULE NAME
5	
10	
	MONTH
15	DAY
	YEAR
20	HOUR
	MINUTE
	MODULE DESCRIPTION
25	
30	
35	
40	
45	
50	

50	
	MODULE DESCRIPTION, CONT.
55	
60	
65	
70	

Figure B-2. Task or Subroutine Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 2 *

RECORD TYPE

PAGE NUMBER

5

SOURCE SEQUENCE NUMBER

10

15

VERSION INFORMATION

20

25

30

35

40

45

50

50

VERSION INFORMATION, CONT.

55

60

65

70

Figure B-3. Listing and Version Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 3 *

	RECORD TYPE
	PROLOGUE COMMENT
5	
10	
15	
20	
25	
30	
35	
40	
45	
50	

50	
	PROLOGUE COMMENT, CONT.
55	
60	
65	
70	

Figure B-4. Prologue Comments

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 4 *

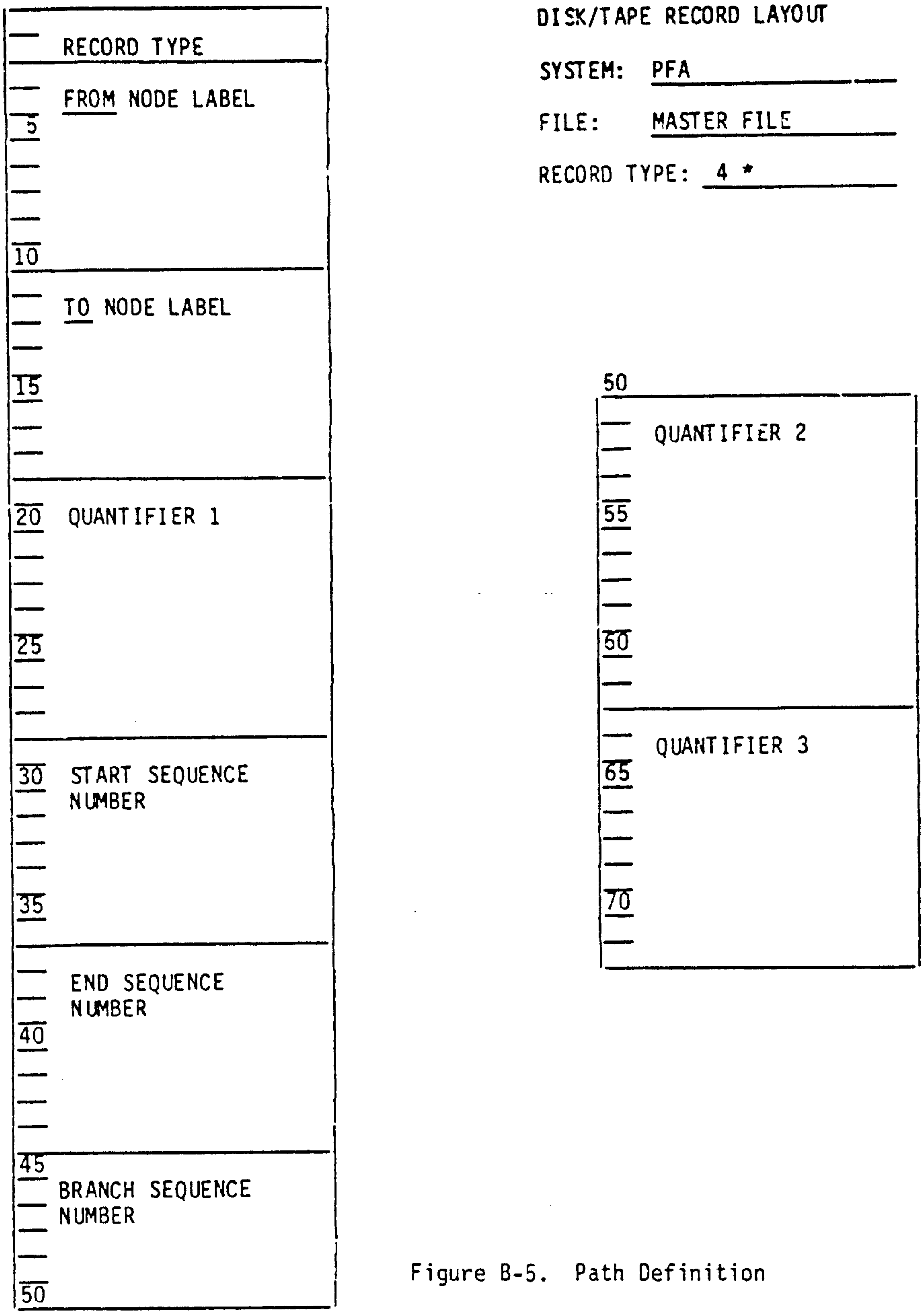


Figure B-5. Path Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 5 *

RECORD TYPE

FROM NODE LABEL

5

10

TO NODE LABEL

15

CONTINUATION FLAG

20

SUBROUTINE CALL LIST

25

30

35

40

45

50

50

SUBROUTINE CALL LIST, CONT.

55

60

65

70

Figure B-6. Subroutine Calls Informacion

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 6 *

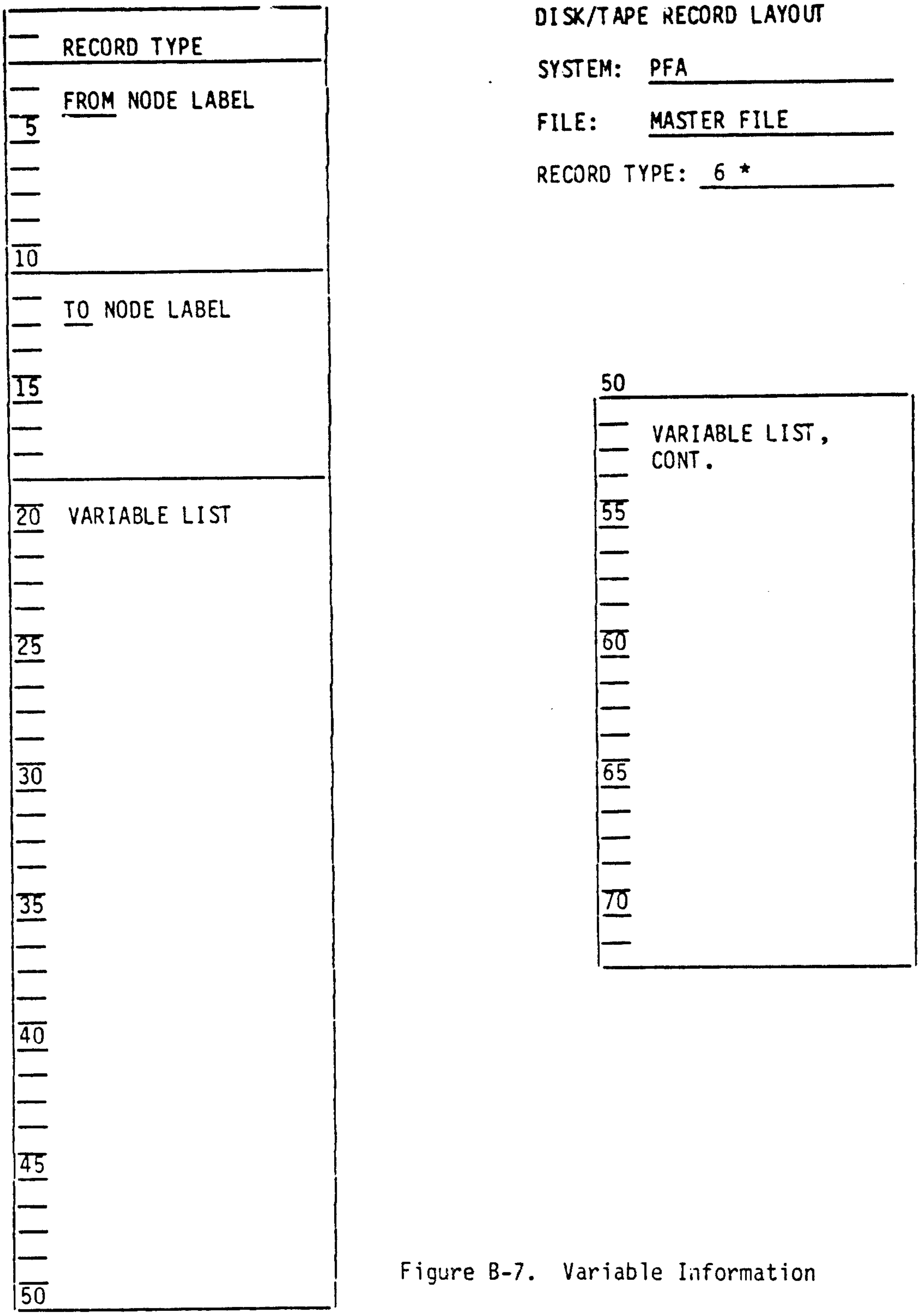


Figure B-7. Variable Information

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 7 *

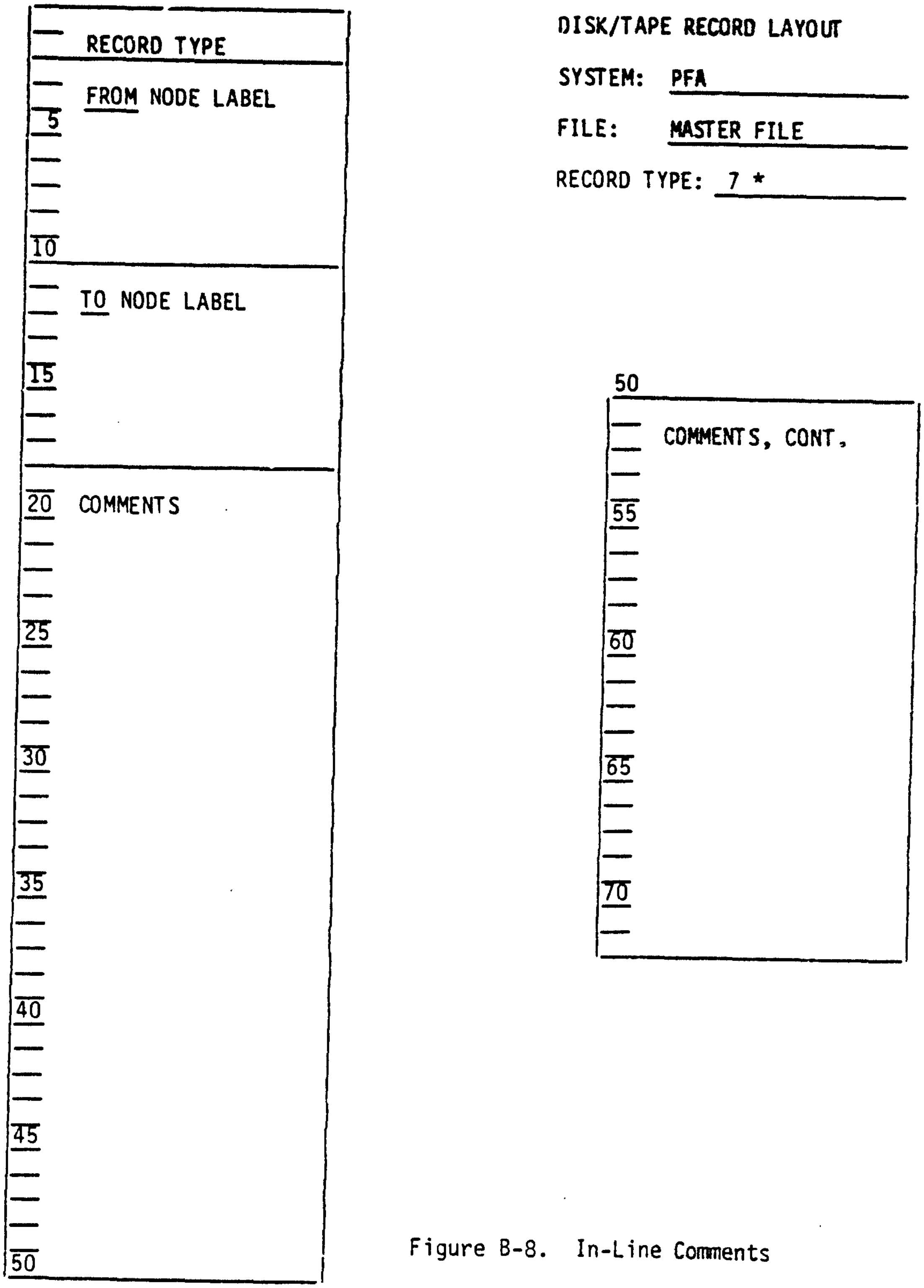


Figure B-8. In-Line Comments

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 8 *

	RECORD TYPE
5	SOFTWARE QUALITY PARAMETER 1
10	SOFTWARE QUALITY PARAMETER 2
15	SOFTWARE QUALITY PARAMETER 3
20	SOFTWARE QUALITY PARAMETER 4
25	SOFTWARE QUALITY PARAMETER 5
30	SOFTWARE QUALITY PARAMETER 6
35	SOFTWARE QUALITY PARAMETER 7
40	SOFTWARE QUALITY PARAMETER 8
45	SOFTWARE QUALITY PARAMETER 9
50	SOFTWARE QUALITY PARAMETER 10

52

55	SOFTWARE QUALITY PARAMETER 11
60	SOFTWARE QUALITY PARAMETER 12
65	SOFTWARE QUALITY PARAMETER 13
70	SOFTWARE QUALITY PARAMETER 14

Figure B-9. Software Quality Information

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 9 *

	RECORD TYPE
	ENTRY POINT NAME
5	
10	
	LINK
15	
20	
	START NODE LABEL
25	
30	
	ENTRY POINT DESCRIPTION
35	
40	
45	
50	

50	
	ENTRY POINT DESCRIPTION, CONT.
55	
60	
65	
70	

Figure B-10. Entry Point Definition

TABLE I. MASTER FILE FIELD DESCRIPTION

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
Record Type	0-9	Denotes record type	RECTYPE (STAMP SYSTRUCT) CD (PATCHANA)	2	'0*' ,'1*'...'9*'
Module Name	0, 1	Module name, truncated if necessary (but must be unique)	TASKNAME (STAMP, SYSTRUCT) MODNAME (SOFTPRO, PATCHANA	10	———
Month	0, 1	Not used		2	1-12
Day	0, 1	Not used		2	1-31
Year	0, 1	Not used		2	0-99
Hour	0, 1	Not used		2	0-23
Minute	0, 1	Not used		2	0-59
Module Description	0, 1	Optional brief description of module	TDESC (STAMP, SYSTRUCT) DESC (SOFTPRO, PATCHANA)	50	———
Page Number	2	Optional page number of start of module in encoder listing	TPAGE (STAMP, SYSTRUCT) PAGE (SOFTPRO, PATCHANA)	5	0, 99999
Source Sequence Number	2	Optional sequence number of start of module in source listing	TSRCSEQ (STAMP, SYSTRUCT) SEQ (SOFTPRO, PATCHANA)	8	0, 99999999
Version Information	2	Optional module version information	VERSION (STAMP, SYSTRUCT, SOFTPRO, PATCHANA)	57	———
Prologue Comment	3	One comment line from prologue	———	70	———

TABLE I. MASTER FILE FIELD DESCRIPTIONS (CONTINUED)

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
From Node Label	4-7	Label giving start point of path segment in program	FROM (STAMP, PATCHANA)	8	Alpha or numeric
To Node Label	4-7	Start label of segment to which control is transferred	TO (STAMP, PATCHANA)	8	Alpha or rumberic
Quantifier 1	4	Optional path	QUANTI (STAMP, PATCHANA)	10	Integer or real number
Start Sequence Number	4	Optional source sequence number of start of path	SEQSTRT (STAMP, PATCHANA)	8	0-99999999
End Sequence Number	4	Optional source sequence number of end of path	SEQEND (STAMP, PATCHANA)	8	0-99999999
Branch Sequence Number	4	Optional source sequence number of segment to which control is transferred	SEQJMP (STAMP, PATCHANA)	8	0-99999999
Quantifier 2	4	Optional path quantifier. Varies from encoder to encoder	QUANT2 (STAMP, PATCHANA)	10	Integer or real number
Quantifier 3	4	Optional path quantifier. Varies from encoder to encoder	QUANT3 (STAMP, PATCHANA)	10	Integer or real number
Continuation Flag	5	Field indicating whether this type 5 card is the continuation of the previous type 5 card	CONT (SYSTRUCT)	1	=0 Nct a continuation >0 continuation

TABLE I. MASTER FILE FIELD DESCRIPTIONS (CONTINUED)

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
Subroutine Call List	5	List of subroutine calls in path segment. Calls are separated by commas.	SUBRTN (STAMP, PATCHANA) ITCL (SYSTRUCT)	53	———
Variable List	6	List of variables appearing in path segment. The variable name is followed by two colons. After the first colon may appear the optional variable type. After the second colon appears: R - if variable referenced M - if variable modified D - if variable defined T - if variable tested	SYMBOL (STAMP, PATCHANA)	54	———
Comment	7	In-line comment from source	COMMENT (STAMP)	54	———
Software Quality Parameters	8	Each field gives the value of some software quality parameter for this module. The actual parameters given vary from encoder to encoder	TYPE8 (SYSTRUCT) VALUES14 (SOFTPRO)	14 fields each of width 5	Integer
Entry Point Name	9	Name of entry point, truncated if necessary	TASKNAME (STAMP) MODNAME (PATCHANA)	10	———
Link	9	Name of module in which entry point appears	LINK (STAMP, PATCHANA)	10	———
Start Node Label	9	Label of entry point start node	STARTNODE (STAMP) STNODE (PATCHANA)	8	———

TABLE 1. MASTER FILE FIELD DESCRIPTIONS (CONTINUED)

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
Entry Point Description	9	Brief description of entry point.	TDESC (STAMP, SYSTRUCT) DESC (PATCHANA)	42	———

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: STANDARDS FILE

IDENTIFIER ;
DESCRIPTION ;
WEIGHTING FACTOR ;
LOWEST PERMITTED VALUE ;
HIGHEST PERMITTED VALUE ;

VALUE OR CALCULATION ;
CODE/DOCUMENTATION INDICATOR ;
STANDARD REFERENCE ;

Figure B-11. Software Standards File Records

TABLE II. STANDARD FILE FIELD DESCRIPTION

NAME	PURPOSE	TAGS (PROGRAMS)	RANGE
Identifier	Unique identification of standard description; maximum of four (4) characters. If the ID is numeric and from 1-14, the description is of a "basic" standard, a standard whose value is calculated by the encoder and stored in the master file. If the ID is any other value, the description is of a "user-defined" standard whose value is calculated in terms of previously defined standards.	IDX (SYSTRUCT) STDLABEL (SOFTPRO)	——
Description	A description of the standard (maximum size of 40).	DESCRIPT (SYSTRUCT) STDDESCR (SOFTPRO)	——
Weighting Factor	Flag indicating whether standard is to be reported or merely calculated.	WEIGHT (SOFTPRO, SYSTRUCT)	=1 Report =0 Do not Report
Lowest Permitted Value	The lower end of the acceptable range for this standard.	MINVAL (SYSTRUCT) LOWVAL (SOFTPRO)	Integer
Highest Permitted Value	The higher end of the acceptable range for this standard.	MAXVAL (SYSTRUCT) HIGHVAL (SOFTPRO)	Integer
Calculation	An expression for calculating the value of user-defined standards. Null for basic standards. The expression is in reverse polish notation and may consist of constants, the operators '+', '-', '*', and '/', and the values of previously defined standards, referenced by the standard's ID in parentheses. For example; a legal expression is '(5), (1) /,'. (See section 3.3.3.5)	GIVEN (SYSTRUCT) STDEXPR (SOFTPRO)	——
Code/Documentation Indicator	Flag indicating that the described standard is a code standard or a documentation standard.	SIGNIFIC (SYSTRUCT) DOCCODE (SOFTPRO)	=C Code =D Documentation
Standard Reference	Reference to the document and paragraph in which the described standard is defined (maximum size of 40).	REFERENCE (SOFTPRO)	——

APPENDIX G

PROGRAM MAINTENANCE MANUAL

PROGRAM FLOW ANALYZEʀ

PROGRAM MAINTENANCE MANUAL

JANUARY 1984

FOREWORD

Ultrasystems Technology, Incorporated, Sierra Vista, Arizona assisted in the preparation of this document under Contract Number DAEA18-83-C-0003.

TABLE OF CONTENTS

VOLUME I

APPENDIXES

VOLUME II

LIST OF FIGURES

LIST OF TABLES

SECTION 1, GENERAL DESCRIPTION

1.1 Purpose of the Program Maintenance Manual. The objective of this Program Maintenance Manual for the Program Flow Analyzer (PFA) system, TECOM Project Number 7-CO-RDO-EP1-004, is to provide the maintenance programmer presonnel with the information necessary to effectively maintain the system.

1.2 Project References. PFA is a software analysis system. It consists of various programs which identify software quality, program structure, and program features which aid or interfere with program maintainability.

PFA is sponsored by the U.S. Army Electronic Proving Ground (USAEPG) at Fort Huachuca.

The following are PFA documents of interest to the maintainer.

USAEPG, Program Flow Analyzer Users Manual, 30 November 1982, UNCLASSIFIED

USAEPG, Program Flow Analyzer Plan, 30 October 1982, UNCLASSIFIED

USAEPG, Methodology Investigation Proposal--Program Flow Analyzer, March 1979, UNCLASSIFIED

USAEPG, Program Flow Analyzer A-Level Specification, 4 April 1980, UNCLASSIFIED.

1.3 Terms and Abbreviations. Terms, definitions, abbreviations, and acronyms are included in appendix A.

SECTION 2. SYSTEM DESCRIPTION

2.1 System Application. PFA is a software tool used for automating the analysis of a software system to identify software quality, program structure, and program maintainability. PFA performs the following functions:

a. Software documentation aid

b. Program structure analysis on the system level

c. Program structure analysis on the module level

d. Software quality analysis

e. Analysis of system configuration changes

f. Software modification analysis on the module level

PFA can automate review of design-level specifications and code, software quality metrics, and comparison of different versions of software which are normally performed manually. PFA generates reports that can be compared to design specifications; this reduces effort needed to trace from code to design specifications. PFA computes and compares software quality metrics to user-selected software quality standards, thereby automating the software quality assessment. PFA compares two versions of software, first an overall comparison and then a comparison of selected modules, and generates reports which identify changes.

2.2 Security and Privacy. The PFA system is unclassified and is currently set up to identify all output as unclassified.

2.3 General Description. The structure of the PFA system is presented in figure 2-1.

2.4 Program Description. The following paragraphs provide a description of the PFA programs: AUTOxxxx, COMPRO, STAMP, SOFTPRO, SYSTRUCT, PATCHANA, and PFALIB. The encoder program (AUTOxxxx) is specific to the language/machine combination being analyzed. Only details applicable to encoders in general are provided here. The remaining PFA programs are generic in nature. HIPO and structure charts for these programs are included in Volume II, Appendix B and C. PFA programs are written in MACROSPITBOL with the exception of SOFTPRO, written in VAX FORTRAN.

2.4.1 Encoder (AUTOxxxx). The encoder is a front-end program to the PFA system, which translates computer/software/language-specific programs into a representation which is stored in a data base called the master file. This master file is then read by other PFA programs that generate reports. The encoder is customized for each application to accommodate the specific machine/language combination of the software being analyzed.

2.4.2 Comment Processor (COMPRO). COMPRO extracts and processes comments for creating documentation. COMPRO reads program source code files that contain comments in a specific format. The information contained in these comments is used to create a master file that can be processed by other PFA programs to generate reports used for program documentation. PFA programs are commented in tnis manner to provide automated HIPO and structure charts for the PFA system.

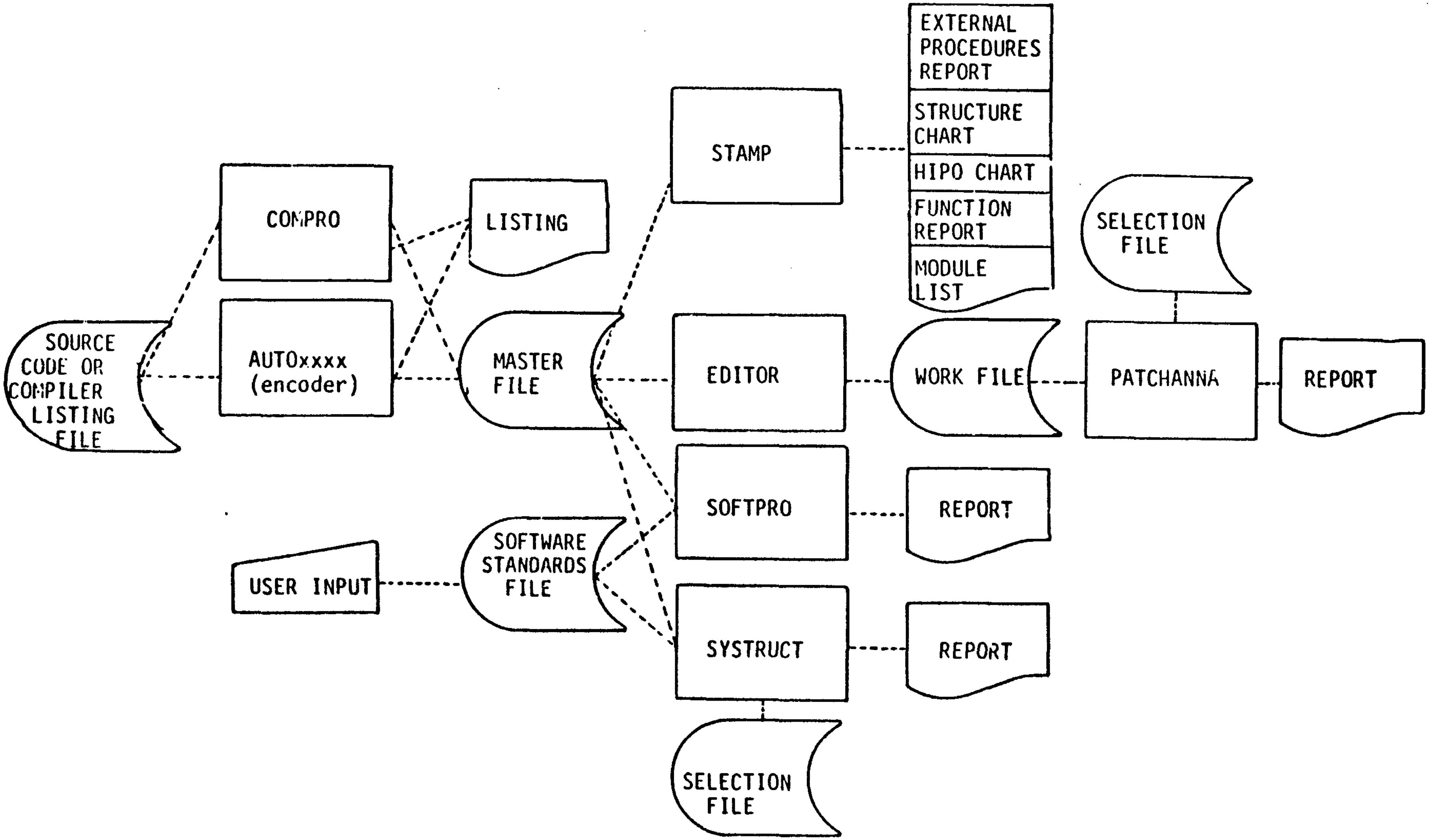


Figure 2-1. Structure of the PFA System

2.4.3 Structure, Timing, Analysis, Modeling Program (STAMP). STAMP provides the program structure analysis function. The STAMP program reads the master file and generates overall reports on the system under analysis to aid in modeling that system and to provide information about the structure of that system. STAMP provides five reports, as follows:

a. The external procedures report lists each module which calls subroutines outside the set of modules being analyzed and the name of the subroutines called.

b. The structure chart shows the module call hierarchy of the system and flags external and recursive subroutines.

c. The function list provides a list of all task-level modules in the system, with the name of the module, a brief description, and version information.

d. The module list contains a directory of the modules being analyzed, arranged in alphabetical order. It also contains an indicator to signal whether the module is task level, subroutine level, or an unused entry point; the top and bottom hierarchical level on which the module is called; a brief module description; the version information; and the page number of the source listing where the module is located.

e. The HIPO chart lists the name of the module, a brief module description, the version information, the page number of the source listing prologue comments, and the variables input and output. The processing description in the HIPO chart includes the path segment structure of the module. Each path segment contains the names of modules called, comments about the path, three quantifiers (discussed below) and the sequence number range of the source statements which make up the path segment.

The quantifiers are user-selected values (determined by the particular encoder used) extracted from the software being analyzed. Examples are execution module timing, source statement counts, and machine instruction counts. Those items which are not available do not prohibit processing. The STAMP program will report on the information available.

2.4.4 Software Profile Program (SOFTPRO). SOFTPRO provides the software quality analysis function. SOFTPRO reads the master file and a software standards file and generates two reports, as follows:

a. The software profile reports for each module list the violations of software standards as defined by the user in the software standards file. The violations state the standard, the value for the software being analyzed, the permitted range of values, and the reference to the regulation from which the standard is derived.

b. The profile summary contains the list of standards, the number and percent of modules meeting each standard, the number and percent of modules violating each standard, and the reference to the regulations containing each standard. Additionally, quality values are summarized for all modules and listed individually for each module.

2.4.5 System Structure Comparator (SYSTRUCT). SYSTRUCT provides analysis of system configuration changes. SYSTRUCT reads two master files representing different versions of the software system under analysis. The SYSTRUCT report lists changes in variables used by each module, changes in software quality parameters (the ones reported by SOFTPRO), and modules which have been added to or deleted from the current version of the software being analyzed. The changes are of two types: those affecting executable code and those affecting code documentation. When changes affect executable code, a list of the modules changed (selection file) is generated for use by the patch analysis program.

2.4.6 Patch Analysis Program (PATCHANA). PATCHANA provides a software patch (source code change) analysis function. PATCHANA reads two master files or subsets of master files representing two versions of the software being analyzed. A selection file which contains a list of modules to be compared is also read. Selection can be made interactively or via the selection file created by SYSTRUCT. The report lists the path segment structure for both versions and indicates (with codes and arrows) the location of differences and the type of change for each version. When structural differences occur, the differences are flagged from their start either through the remainder of the module or to the point where the structures of the two versions are congruent. PATCHANA finds differences in structure, subroutine calls, use of variables, and the three quantifiers.

2.4.7 PFA Library Routines (PFALIB). The PFA library routines are MACROSPITBOL routines that are used by the various PFA programs. The following routines constitute PFALIB:

CENTER returns a given string centered within a string of given length.

CLS clears the terminal (user's) screen.

DECR decrements the argument by one.

DTOO is used to convert decimal numbers into octal numbers.

ENDOUT prints the 'UNCLASSIFIED' caveat on the last page.

ENTER returns the next available channel number.

FIELD1 extracts a string of characters of a given length from a larger string of characters, starting at a given character position. It also trims leading and trailing blanks from the string it returns.

INCR increments the argument by one.

INIOUT performs initialization for OUTOUT.

ITMCMP determines if an item is in a list.

LIST returns an item at a specified index. If the given index is greater than the index of the last item in the list, the last item is returned.

LSTADD adds a given item to the end of the list.

LSTDLN deletes an item at a given index from a list.

LSTDLS deletes all occurrences of a given item from a list.

LSTFND returns the index of a given item in a list.

LSTGET returns the item at a given index from a list.

LSTINN inserts an item at a given index in a list.

LSTINS inserts an item before another given item in a list.

LSTLEN returns the number of items in a list.

MAX returns the maximum of two real or integer numbers.

MIN returns the minimum of two real or integer numbers.

NODUP deletes duplicate entries from a list.

OCTBIN returns a binary representation (ones and zeros) of a given octal number.

OTOD returns a decimal representation of a given octal number.

OUTOUT outputs a given line and provides the 'UNCLASSIFIED' caveat, page numbering, and header at page breaks.

PAD returns a string of blanks of given length.

PRETRM trims leading blanks from a given string.

REWIND rewinds a file on a given channel and releases variables tied to that channel.

ROUND returns a given real number rounded to two decimal places. It does not affect integer numbers.

STROUT takes as many as the given number of characters, as delimited by a comma, from a given string and places them on the returned string. The input string is shortened by the number of characters in the returned string.

SWAP swaps the contents of two variables.

SECTION 3. ENVIRONMENT

3.1 Equipment Environment. PFA is currently implemented on a DEC VAX/VMS, model 11/780. PFA software is resident on disk; software to be analyzed is input via nine-track tape drives.

3.2 Support Software. PFA currently requires the following DEC software:

a. VMS, Version 3.0 or later

b. VAX FORTRAN

The following software from other sources is also required:

MACROSPITBOL, available from DEWAR Information Systems, Inc., 221 West Lake Street, Oak Park, IL 60302

3.3 Data Base. PFA uses two data files, the master file and the standards file. The master file contains condensed/summary information on the program to be analyzed. It is used by STAMP, SOFTPRO, SYSTRUCT, and PATCHANA to analyze and report on program structure, maintainability, and software quality. The master file may be edited by using the system editor.

The standards file contains information on software standards. It is used by SOFTPRO to calculate and report standards violations and by SYSTRUCT to identify software quality measures which have changed between two versions of the software being analyzed.

3.3.1 General Characteristics. The following are the general characteristics of the data base files:

3.3.1.1 Master File. The master file is created by system-specific encoders (AUTOxxxx and COMPRO). The master file filename is specified by the user at run time. The user may specify any filename acceptable to the system, but, by convention, the file extension is MST. The master file, or a subset, is used by STAMP, SOFTPRO, SYSTRUCT, and PATCHANA to analyze and report on program structure, maintainability, and software quality. Additionally, the master file may be edited. The master file is read-only to all programs except the editor, AUTOxxxx, and COMPRO. The master file is an ASCII sequential file resident on disk storage. Records are fixed format, delimited by a carriage return/linefeed sequence, with a data length of 72 characters. The amount of storage required varies with the application.

3.3.1.2 Standards File. The standards file is created in two parts. First, the developer of the encoder creates a file containing descriptions of the basic standards, for which values are output by the encoder. Then, before run time, the user adds descriptions of standards which are user defined in terms of the basic standards. The standards file filename is specified by the user at run time. The user may specify any name acceptable to the system but, by convention, the file extension is STD. The standards file is used by SOFTPRO to calculate and report standards violations and by SYSTRUCT to identify software quality measures which have changed between two versions of the program being analyzed. The standards file is read-only to all PFA programs.

The standards file is a free format, ASCII, sequential file maintained on disk storage. The amount of storage required varies with the application.

3.3.2 Organization and Detailed Description. The following is a detailed description of the database files.

3.3.2.1 Master File.

a. Layout. Figures 3-1 through 3-10 show the layout of the master file. Currently there are eleven record types in the files (a header type and ten data types). The eleven record types contain the following information:

Type	Information
-	Header Record
0	Library Routine Definition
1	Task or Subroutine Definition
2	Listing and Version Information
3	Prologue Comments
4	Path Definition
5	Subroutine Calls Information
6	Variable Information
7	In-Line Comments
8	Software Quality Information
9	Entry Point Definition

b. Groupings and Order. Within the master file, all the records for a module are grouped together. Within each module group, the records have the following order:

(1) A header record of the form: "#-H- modname", left-justified.

(2) A type 0 or 1 record, as appropriate.

(3) A type 2 record and, optionally, type 3 records. The type 2 record may appear anywhere within this group.

(4) The path information records are grouped together by path and include type 4, 5, 6 and 7 records. Each group defines a path. The definition of a path includes the specification of a from node, or start point, for the path. The groups are ordered so that the from nodes specified by the groups appear in the master file in the same order as the from nodes appear in the program listing. The records within the path groups are ordered as follows: The optional type 4 record must appear first, followed by optional type 5, 6 and 7 records, which may be intermixed.

(5) Optional type 8 records, in order.

(6) Optional type 9 records which may be intermixed with type 8 records and/or path groups.

c. Fields. See Figure 3-1 through 3-10 for the layout of the fields described in Table 3-I.

3.3.2.2 Standards File.

The software standards file is used for comparing the software being analyzed to specific standards.

a. Layout. Each record in the standards file contains eight fields, each terminated by a semicolon.

b. Groupings and Order. The records in the standards file may be in any order.

c. Fields. A sample standards file is presented in figure 3-11. The fields are defined as follows:

(1) Identifier. The identifier has two ranges:

(a) 1 to 14, basic standard taken directly from the corresponding field on the master file type 8 record.

(b) Any other value.

Range (a) specifies which item of the software quality record the standard record describes. Range (b) provides scratch records for calculations or contains identification and calculations for derived or calculated software metrics.

(2) String description. This is a brief description of the standard (maximum of 40 characters) stated as the requirement needed (value) to meet/pass the standard, for example: McCabe's complexity $\leq$ 10 (meets standard)

(3) Weighting factor. This is an integer, zero or greater, which gives a relative weight to each standard. Zero means the record is for calculation only and is not reported. Numbers greater than zero rate the relative severity of violating the standard; the greater the number, the more severe the consequences of the violation.

(4) Lowest permitted value. If the value supplied (or calculated) for the standard record is less than the lowest permitted value, the standard is violated.

(5) Highest permitted value. If the value supplied (or calculated) for the standard record is greater than the highest permitted value, the standard is violated.

(6) Value or calculation. If the identifier is in the range (a) above, then this field is left blank; otherwise, the field contains an expres-

sion used for calculations. The expression must be in reverse Polish notation with a comma following each term of the expression. Terms which are enclosed in parentheses are identifiers which are replaced by a value or calculation from the standard record indicated by that identifier. Terms which are numbers are used as constants in the calculations. The four arithmetic operators permitted are +, -, *, /.

(7) Code/documentation indicator. This indicator is "C" if the standard affects source code specific to a programming language or "D" if the standard affects program documentation contained in comments.

(8) Standard reference. This field refers to the document where the standard is found.

See figure 3-12 for the layout of the fields described in Table 3-II. The sizes of all fields are variable. Null fields are indicated by two consecutive semicolons.

d. Provisions for Expansion. There are provisions for 14 basic standards with up to 30 total.

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 0 *

	RECORD TYPE
	MODULE NAME
5	
10	
	MONTH
15	DAY
	YEAR
20	HOUk
	MINUTE
	MODULE DESCRIPTION
25	
30	
35	
40	
45	
50	

50	
	MODULE DESCRIPTION, CONT.
55	
60	
65	
70	

Figure 3-1. Library Routine Definition

RECORD TYPE

MODULE NAME

5

10

MONTH

15

DAY

YEAR

20 HOUR

MINUTE

MODULE DESCRIPTION

25

30

35

40

45

50

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 1 *

50

MODULE DESCRIPTION, CONT.

55

60

65

70

Figure 3-2. Task or Subroutine Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 2 *

	RECORD TYPE
	PAGE NUMBER
5	
	SOURCE SEQUENCE NUMBER
10	
15	
	VERSION INFORMATION
20	
25	
30	
35	
40	
45	
50	

50	
	VERSION INFORMATION, CONT.
55	
60	
65	
70	

Figure 3-3. Listing and Version Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 3 *

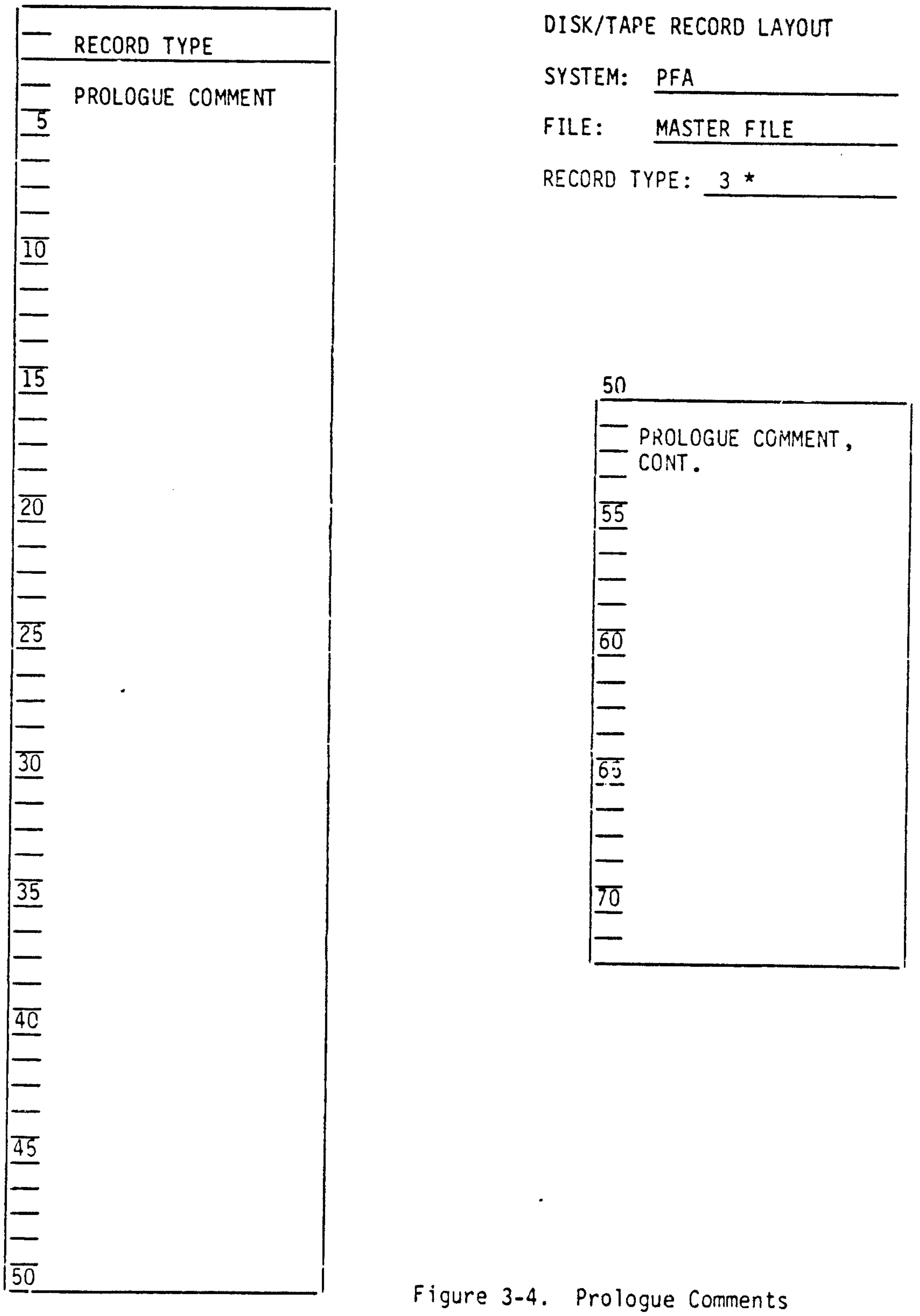


Figure 3-4. Prologue Comments

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 4 *

RECORD TYPE

FROM NODE LABEL
5
10

TO NODE LABEL
15

20 QUANTIFIER 1
25

30 START SEQUENCE NUMBER
35

END SEQUENCE NUMBER
40

45 BRANCH SEQUENCE NUMBER
50

50

QUANTIFIER 2
55
60

QUANTIFIER 3
65
70

Figure 3-5. Path Definition

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 5 *

	RECORD TYPE
	FROM NODE LABEL
5	
10	
	TO NODE LABEL
15	
	CONTINUATION FLAG
20	SUBROUTINE CALL LIST
25	
30	
35	
40	
45	
50	

50	
	SUBROUTINE CALL LIST, CONT.
55	
60	
65	
70	

Figure 3-6. Subroutine Calls Information

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 6 *

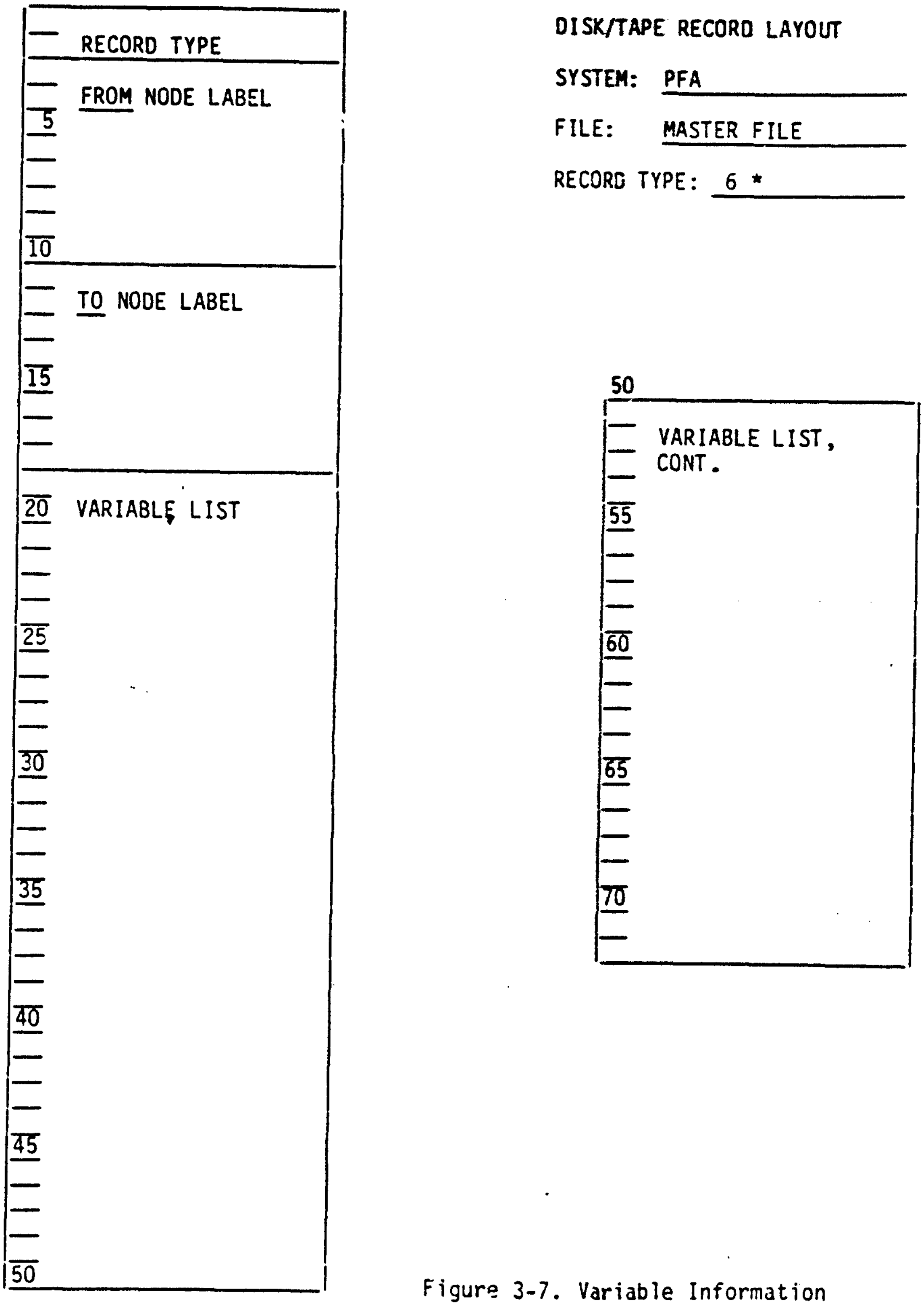


Figure 3-7. Variable Information

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 7 *

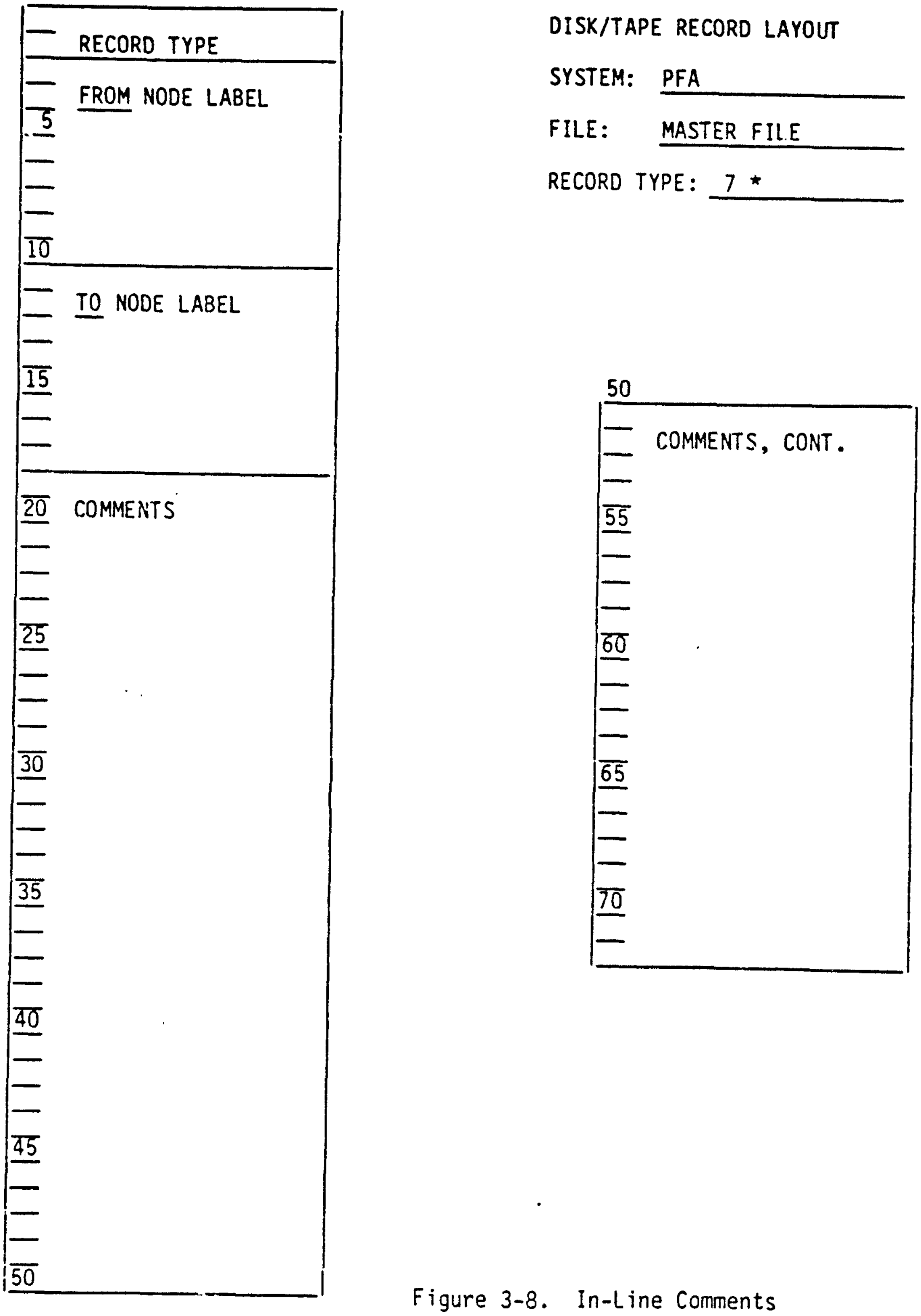


Figure 3-8. In-Line Comments

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 8 *

	RECORD TYPE
5	SOFTWARE QUALITY PARAMETER 1
10	SOFTWARE QUALITY PARAMETER 2
15	SOFTWARE QUALITY PARAMETER 3
20	SOFTWARE QUALITY PARAMETER 4
25	SOFTWARE QUALITY PARAMETER 5
30	SOFTWARE QUALITY PARAMETER 6
35	SOFTWARE QUALITY PARAMETER 7
40	SOFTWARE QUALITY PARAMETER 8
45	SOFTWARE QUALITY PARAMETER 9
50	SOFTWARE QUALITY PARAMETER 10

52

55	SOFTWARE QUALITY PARAMETER 11
60	SOFTWARE QUALITY PARAMETER 12
65	SOFTWARE QUALITY PARAMETER 13
70	SOFTWARE QUALITY PARAMETER 14

Figure 3-9. Software Quality Information

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: MASTER FILE

RECORD TYPE: 9 *

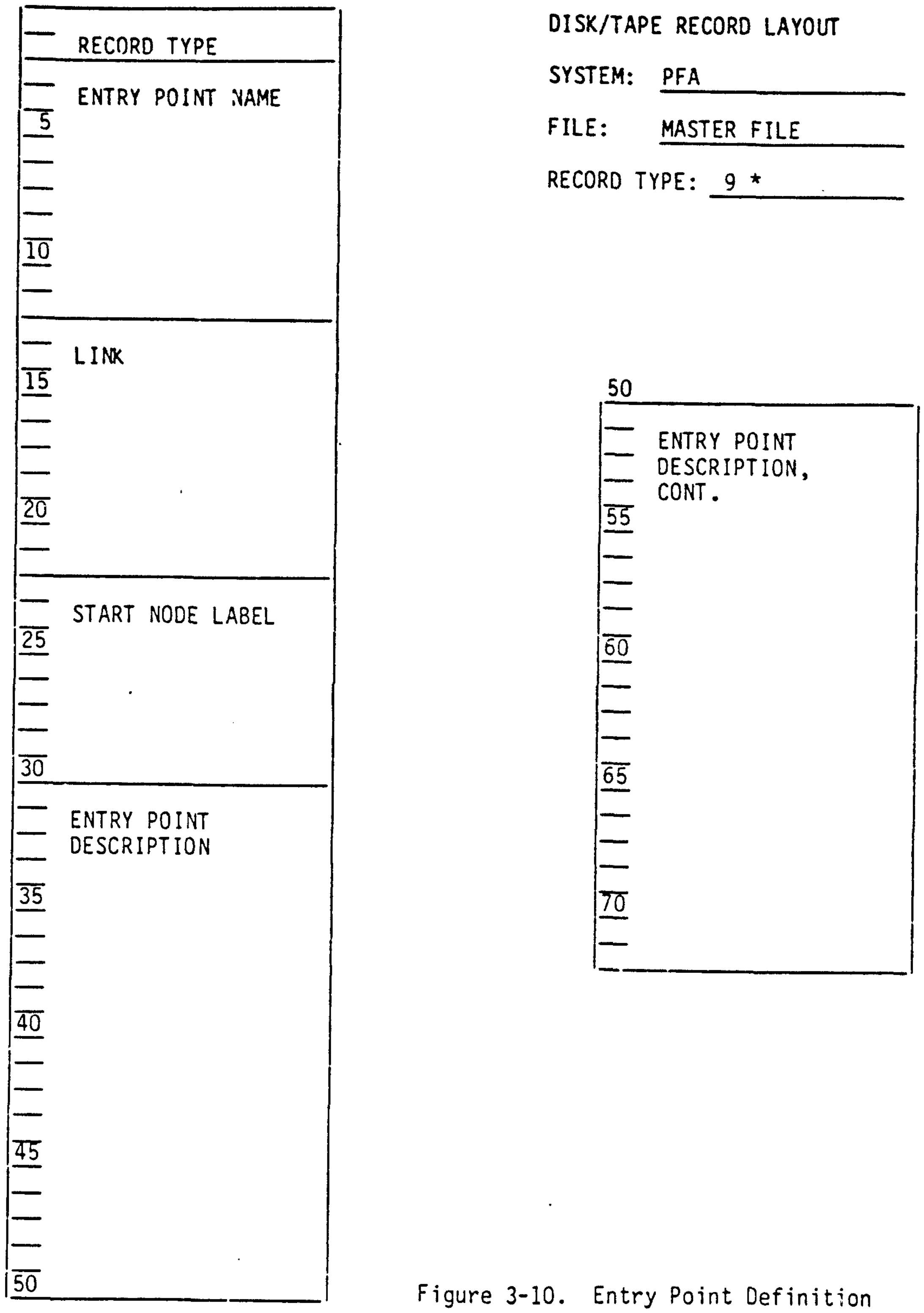


Figure 3-10. Entry Point Definition

TABLE 3-I. MASTER FILE FIELD DESCRIPTION

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
Record Type	0-9	Denotes record type	RECTYPE (STAMP, SYSTRUCT) CD (PATCHANA)	2	'0*' ,'1*'...'9*'
Module Name	0, 1	Module name, truncated if necessary (but must be unique)	TASKNAME (STAMP, SYSTRUCT) MODNAME (SOFTPRO, PATCHANA	10	——
Month	0, 1	Not used		2	1-12
Day	0, 1	Not used		2	1-31
Year	0, 1	Not used		2	0-99
Hour	0, 1	Not used		2	0-23
Minute	0, 1	Not used		2	0-59
Module Description	0, 1	Optional brief description of module	TDESC (STAMP, SYSTRUCT) DESC (SOFTPRO, PATCHANA)	50	——
Page Number	2	Optional page number of start of module in encoder listing	TPAGE (STAMP, SYSTRUCT) PAGE (SOFTPRO, PATCHANA)	5	0, 99999
Source Sequence Number	2	Optional sequence number of start of module in source listing	TSRCSEQ (STAMP, SYSTRUCT) SEQ (SOFTPRO, PATCHANA)	8	0, 99999999
Version Information	2	Optional module version information	VERSION (STAMP, SYSTRUCT, SOFTPRO, PATCHANA)	57	——
Prologue Comment	3	One comment line from prologue	——————	70	——

TABLE 3-I. MASTER FILE FIELD DESCRIPTIONS (CONTINUED)

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
From Node Label	4-7	Label giving start point of path segment in program	FROM (STAMP, PATCHANA)	8	Alpha or numeric
To Node Label	4-7	Start label of segment to which control is transferred	TO (STAMP, PATCHANA)	8	Alpha or numberic
Quantifier 1	4	Optional path	QUANTI (STAMP, PATCHANA)	10	Integer or real number
Start Sequence Number	4	Optional source sequence number of start of path	SEQSTRT (STAMP, PATCHANA)	8	0-99999999
End Sequence Number	4	Optional source sequence number of end of path	SEQEND (STAMP, PATCHANA)	8	0-99999999
Branch Sequence Number	4	Optional source sequence number of segment to which control is transferred	SEQJMP (STAMP, PATCHANA)	8	0-99999999
Quantifier 2	4	Optional path quantifier. Varies from encoder to encoder	QUANT2 (STAMP, PATCHANA)	10	Integer or real number
Quantifier 3	4	Optional path quantifier. Varies from encoder to encoder	QUANT3 (STAMP, PATCHANA)	10	Integer or real number
Continuation Flag	5	Field indicating whether this type 5 card is the continuation of the previous type 5 card	CONT (SYSTRUCT)	1	=0 Not a continuation >0 continuation

TABLE 3-I. MASTER FILE FIELD DESCRIPTIONS (CONTINUED)

NAME	RECORD #	PURPOSE	TAGS (PROGRAMS)	FIELD SIZE	RANGE
Subroutine Call List	5	List of subroutine calls in path segment. Calls are separated by commas.	SUBRTN (STAMP, PATCHANA) ITCL (SYSTRUCT)	53	——
Variable List	6	List of variables appearing in path segment. The variable name is followed by two colons. After the first colon may appear the optional variable type. After the second colon appears: R - if variable referenced M - if variable modified D - if variable defined T - if variable tested	SYMBOL (STAMP, PATCHANA)	54	——
Comment	7	In-line comment from source	COMMENT (STAMP)	54	——
Software Quality Parameters	8	Each field gives the value of some software quality parameter for this module. The actual parameters given vary from encoder to encoder	TYPE8 (SYSTRUCT) VALUES14 (SOFTPRO)	14 fields each of width 5	Integer
Entry Point Name	9	Name of entry point, truncated if necessary	TASKNAME (STAMP) MODNAME (PATCHANA)	10	——
Link	9	Name of module in which entry point appears	LINK (STAMP, PATCHANA)	10	——
Start Node Label	9	Label of entry point start node	STARTNODE (STAMP) STNODE (PATCHANA)	8	——

```
1,COMMENT LINES IN PROLOG > 30,2;30;9999,;D,MIL-STD-1679, 5.4.4.1;
A,COMMENT LINES IN PROLOG > 60,2;60;9999:(1),,D,EPG, DOCUMENTATION;
B,COMMENT LINES IN PROLOG > 120,2;120,9999,(1),;D;EPG, DOCUMENTATION,
2,NUMBER OF NON-BLANK COMMENT LINES,0;0,9999,;D;EPG, DOCUMENTATION,
3,NUMBER OF NON-COMMENT LINES;0;0;9999,;D;;
4,NUMBER OF EXECUTABLE STATEMENTS < 200;2,0;200;;D;MIL-STD-1679, 5.3.7;
5,NUMBER OF STATEMENTS WITH EMBEDDED COMMENTS,0;0;9999,;D,EPG, DOCUMENTATION;
C,50% CODE COMMENTED,2;50,99999,(2),(5),+,100,*,(4),/,;D,EPG, DOCUMENTATION,
D,75% CODE COMMENTED,2;75,99999,(2),(5),+,100,*,(4),/,,D;EPG, DOCUMENTATION,
E,100% CODE COMMENTED,2;100,99999,(2),(5),+,100,*,(4),/,,D,MILSTD-1679,5.4.4.2;
6,NUMBER OF ENTRY POINTS = 1,1;0;1,;C;MIL-STD-1679, 5.3.3,
7,HALLSTEADS COMPLEXITY < 5;0;1;5,,C;EPG, COMPLEXITY;
8,KNOTS COMPLEXITY < 3,0;0,3,,S;EPG, STRUCTURE,
9,MCCABE'S COMPLEXITY < 10;1;1;9,;C;EPG, COMPLEXITY,
F,MCCABE'S COMPLEXITY < 20,1;1;19;(9),;C;EPG, COMPLEXITY;
G,MCCABE'S COMPLEXITY < 30,1;1;29;(9),;C;EPG, COMPLEXITY;
H,NUMBER OF LINES < 500,2;1,500,(1),(2),+,(3),+,;C,EPG, COMPLEXITY;
I,NUMBER OF DECLARATION STATEMENTS < 25;1,0;25,(3),(4),-,;C,EPG, COMPLEXITY;
J,NUMBER OF DECLARATION STATEMENTS < 50;1;0;50;(3),(4),-,;C;EPG, COMPLEXI ,;
K,NUMBER OF DECLARATION STATEMENTS <100;1;0;100;(3),(4),-,;C,EPG, COMPLEXITY;
10,FUTURE DEVELOPMENT (TOP DOWN STRUCTURE);0;1;3;;S,EPG, STRUCTURE;
11;FUTURE DEV.;0;1;100,;S;EPG, NEW METRIC;
```

Figure 3-11. Software Standards File

DISK/TAPE RECORD LAYOUT

SYSTEM: PFA

FILE: STANDARDS FILE

IDENTIFIER
;

DESCRIPTION
;

WEIGHTING FACTOR
;

LOWEST PERMITTED VALUE
;

HIGHEST PERMITTED VALUE
;

VALUE OR CALCULATION
;

CODE/DOCUMENTATION INDICATOR
;

STANDARD REFERENCE
;

Figure 3-12. Software Standards File Records

TABLE 3-II. STANDARD FILE FIELD DESCRIPTION

NAME	PURPOSE	TAGS (PROGRAMS)	RANGE
Identifier	Unique identification of standard description; maximum of four (4) characters. If the ID is numeric and from 1-14, the description is of a "basic" standard, a standard whose value is calculated by the encoder and stored in the master file. If the ID is any other value, the description is of a "user-defined" standard whose value is calculated in terms of previously defined standards.	IDX (SYSTRUCT) STDLABEL (SOFTPRO)	———
Description	A description of the standard (maximum size of 40).	DESCRIPT (SYSTRUCT) STDDESCR (SOFTPRO)	———
Weighting Factor	Flag indicating whether standard is to be reported or merely calculated.	WEIGHT (SOFTPRO, SYSTRUCT)	=1 Report =0 Do not Report
Lowest Permitted Value	The lower end of the acceptable range for this standard.	MINVAL (SYSTRUCT) LOWVAL (SOFTPRO)	Integer
Highest Permitted Value	The higher end of the acceptable range for this standard.	MAXVAL (SYSTRUCT) HIGHVAL (SOFTPRO)	Integer
Calculation	An expression for calculating the value of user-defined standards. Null for basic standards. The expression is in reverse polish notation and may consist of constants, the operators '+', '-', '*', and '/', and the values of previously defined standards, referenced by the standard's ID in parentheses. For example; a legal expression is '(5), (1) /,'. (See section 3.3.3.5)	GIVEN (SYSTRUCT) STDEXPR (SOFTPRO)	———
Code/Documentation Indicator	Flag indicating that the described standard is a code standard or a documentation standard.	SIGNIFIC (SYSTRUCT) DOCCODE (SOFTPRO)	=C Code =D Documentation
Standard Reference	Reference to the document and paragraph in which the described standard is defined (maximum size of 40).	REFERENCE (SOFTPRO)	———

SECTION 4. PROGRAM MAINTENANCE PROCEDURES

4.1 Conventions.

4.1.1 Naming Conventions. The following conventions were used for the assignment of mnemonics.

a. Counters usually begin with "N" or contain "NUM".

b. The mnemonics "RL" and "RP" are used for report file line and page count, respectively.

c. The variable "TRUE" is set to 1; "FALSE" is set to 0.

d. Mnemonics which specify input/output channel numbers end with "CHNL" or "CHNNL".

e. Mnemonics which have to do with batch processing mode start with "BAT".

f. Variables used in the reoort file interface may contain "REP".

g. Variables and subroutine names containing "STD" are used in processing standards records.

h. Variables and subroutine names containing "MOD" are used in module processing.

i. Variables and subroutine names containing "OUT" are used in output processing.

j. Variables and subroutine names containing "MST" are used in master file processing or to indicate that data come from a master file.

k. Variables and subroutine names containing "STK" or "STAK" are used for stacks.

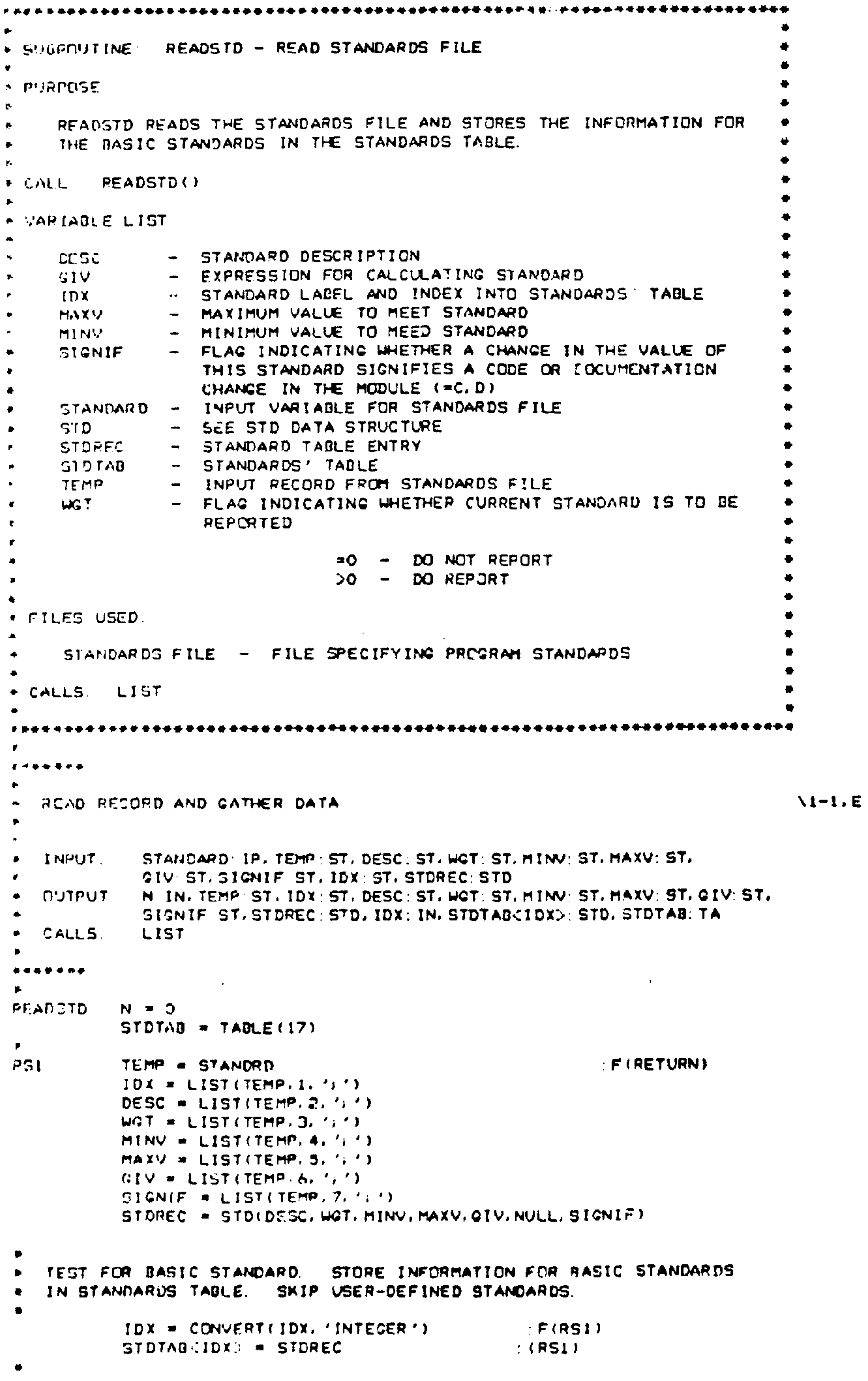
l. Variables and subroutine names containing "NEW" refer to a new version.

m. Variables and subroutine names containing "OLD" refer to an old version.

n. Variables containing "TTY" (e.g., TTY, TTYIN, TTYNOCR) refer to user terminal.

o. Variables containing "ARND" or "EXT" are exit points for local control structures.

4.1.2. Commenting Conventions. Figure 4-1 is an example of the conventions used in PFA source. In general, all comments and code for a module must be grouped together within the source file. A description of the conventions used within a module follows:



```
**********************************************************************
*                                                                    *
* SUBROUTINE   READSTD - READ STANDARDS FILE                         *
*                                                                    *
* PURPOSE                                                            *
*                                                                    *
*    READSTD READS THE STANDARDS FILE AND STORES THE INFORMATION FOR *
*    THE BASIC STANDARDS IN THE STANDARDS TABLE.                     *
*                                                                    *
* CALL   READSTD()                                                   *
*                                                                    *
* VARIABLE LIST                                                      *
*                                                                    *
*    DESC      -  STANDARD DESCRIPTION                               *
*    QIV       -  EXPRESSION FOR CALCULATING STANDARD                *
*    IDX       -  STANDARD LABEL AND INDEX INTO STANDARDS' TABLE     *
*    MAXV      -  MAXIMUM VALUE TO MEET STANDARD                     *
*    MINV      -  MINIMUM VALUE TO MEED STANDARD                     *
*    SIGNIF    -  FLAG INDICATING WHETHER A CHANGE IN THE VALUE OF   *
*                 THIS STANDARD SIGNIFIES A CODE OR DOCUMENTATION    *
*                 CHANGE IN THE MODULE (=C,D)                        *
*    STANDARD  -  INPUT VARIABLE FOR STANDARDS FILE                  *
*    STD       -  SEE STD DATA STRUCTURE                             *
*    STDREC    -  STANDARD TABLE ENTRY                               *
*    STDTAB    -  STANDARDS' TABLE                                   *
*    TEMP      -  INPUT RECORD FROM STANDARDS FILE                   *
*    WGT       -  FLAG INDICATING WHETHER CURRENT STANDARD IS TO BE  *
*                 REPORTED                                           *
*                                                                    *
*                                =0  -  DO NOT REPORT                *
*                                >0  -  DO REPORT                    *
*                                                                    *
* FILES USED.                                                        *
*                                                                    *
*    STANDARDS FILE  -  FILE SPECIFYING PROGRAM STANDARDS            *
*                                                                    *
* CALLS.  LIST                                                       *
*                                                                    *
**********************************************************************
*
*******
*
*  READ RECORD AND GATHER DATA                                            \1-1.E
*
*
*  INPUT.     STANDARD: IP, TEMP: ST, DESC: ST, WGT: ST, MINV: ST, MAXV: ST,
*             QIV: ST, SIGNIF: ST, IDX: ST, STDREC: STD
*  OUTPUT     N: IN, TEMP: ST, IDX: ST, DESC: ST, WGT: ST, MINV: ST, MAXV: ST, QIV: ST,
*             SIGNIF: ST, STDREC: STD, IDX: IN, STDTAB<IDX>: STD, STDTAB: TA
*  CALLS.     LIST
*
*******
*
READSTD   N = 0
          STDTAB = TABLE(17)
*
RS1       TEMP = STANDRD                                  :F(RETURN)
          IDX = LIST(TEMP,1,';')
          DESC = LIST(TEMP,2,';')
          WGT = LIST(TEMP,3,';')
          MINV = LIST(TEMP,4,';')
          MAXV = LIST(TEMP,5,';')
          QIV = LIST(TEMP,6,';')
          SIGNIF = LIST(TEMP,7,';')
          STDREC = STD(DESC,WGT,MINV,MAXV,QIV,NULL,SIGNIF)

*
*  TEST FOR BASIC STANDARD.  STORE INFORMATION FOR BASIC STANDARDS
*  IN STANDARDS TABLE.  SKIP USER-DEFINED STANDARDS.
*
          IDX = CONVERT(IDX,'INTEGER')             :F(RS1)
          STDTAB<IDX> = STDREC                     :(RS1)
*
```

Figure 4-1. A Sample PFA-Commented Source Program

a. Prologues

(1) Each module has a prologue.

(2) The prologue is delimited by asterisks. This means that the prologue must be preceded and followed by a line consisting only of a string of asterisks and that each line in the prologue must begin and end with an asterisk.

(3) Every prologue contains as the first non-blank line:

TASK

SUBROUTINE : <name> - <description>

LIBRARY ROUTINE

where, if the module has multiple entry points, name is the name of the first entry point.

(4) Following item 3 is the module description:

AUTHOR:

DATE STARTED:

DATE LAST MODIFIED:

PURPOSE: <function of module>

DESCRIPTION: <general design description>

CALL:

PARAMETER LIST: <parameters and descriptions>

USER-DEFINED DATA STRUCTURES: <structures, fields and descriptions>

VARIABLE LIST: <variables and descriptions>

FILES USED: <file names and descriptions>

CALLS:

LIMITATIONS:

ERRORS:

as there are appropriate data to include.

(5) For each additional entry point the module contains, (4) above is followed by:

ENTRY POINT: <name> - <description>

and again, as many of the items in (4) above as appropriate.

b. In-Line Comments

(1) All in-line comments begin with an asterisk and are separated from code by strings of asterisks.

(2) In-line comments do not contain ' '.

(3) The in-line comments may contain pseudo path segment specifications. These specifications document the flow of control in a module and are used by COMPRO, the comment processor, and STAMP to create HIPO charts. If used, the path segment specifications have the following format and meaning:

(a) The specification for a path segment begins on the first comment to be included in the segment.

(b) The specification is the rightmost item on the comment line.

(c) The specification is of the form:
[m-][n1,n2,...][E]

where m is the optional node label for the start of the path segment, ni are the optional node labels to which the segment branches, and E indicates branch to exit. If the from node label is null, the from node label will be assumed to be the current node count for this module. If the to node labels (including E) are null, the to node label will be assumed to be the current node count for this module plus one. If this path segment is an entry point for the module but is not the first entry point to the module, m must be the name of the entry point.

(d) The last path specification in a module has a to node specified.

(4) All referenced and modified variables appearing in a pseudo path segment are listed in comments within the path segment for use by COMPRO and STAMP. If there is no path segment specification, all referenced and modified variables for the module are listed. Comments listing referenced or modified variables have the respective formats:

```
INPUT: var1[:type],var2[:type],...vari[:type],

                    .

                    .

                    .

            ...varn[:type]
```

```
OUTPUT:  var1[:type],var2[:type],...vari[:type],

                        .

                        .

                        .

                   ...varn[:type]
```

where type is optional and is an abbreviation for the variable type. In the case of SNOBOL the types are abbreviated as follows:

TYPE	ABBREVIATION
String	ST
Integer	IN
Real Number	RE
Pattern Structure	PA
Array	AR
Table	TA
Created Name	NA
Unevaluated Expression	UN
Object Code	CO
Programmer-defined	Data type name
External	EX
Input	IP
Output	OP

Several lists may be included in one path segment.

(5) All subroutines called in a pseudo path segment are listed in comments within the path segment for use by COMPRO and STAMP. If there is no path segment specification, all subroutines called in the module are listed. Comments listing the calls have the format:

```
CALLS:  call1, call2...calli,

              .

              .

              .
```

...calln

Several lists may be included in one path segment.

4.2 Verification Procedures. Large master files should be used to test the performance of the PFA programs. Master files should contain all possible inputs which affect the PFA program being tested. Master files for testing can be generated by running an encoder on a representative sample of software. The resulting master file can then be edited to add the necessary features for the test.

4.3 Error Conditions.

4.3.1 COMPRO. COMPRO indicates the following errors:

a. Message: "ERR10: ERROR PROCESSING LINE IN MODULE _____"

"<line>"

This means that COMPRO has found a line in the PFA source with an unexpected format. The message indicates an error within COMPRO, and processing terminates.

b. Message: "ERR20: UNEXPECTED END OF FILE IN MODULE _____"

This means that COMPRO has found an end of file while reading a multiline list. The message indicates an error in the PFA source. Processing terminates.

4.3.2 STAMP AND PATCHANA. These programs display the following:

Message: "BAD MAP FOR _____"

This means that there is a to label without a corresponding from label or that the from or to label is null for a path datum in the master file. This results from incomplete path structure generated by the encoder or from dead code in the source code being analyzed.

4.3.3 SOFTPRO. SOFTPRO has the following error conditions:

a. Message: "STANDARD REFERENCE IN STANDARD LINE ____" IS NOT FOUND

This means that a standard reference in the calculation field of the indicated line of the standards file cannot be found.

b. Message: "IN STANDARDS FILE, ERROR IN SPECIFYING CALCULATION FOR STANDARD LINE ____"

This means that the calculation field of the indicated line of the standards file is in error.

c. Message: "IN THE STANDARD FILE, LINE ____ SPECIFIES A CALCULATION FOR A BASIC STANDARD"

This means that for the indicated line in the standards file, the standard number is between 1 and 14, indicating that it is a basic standard, but the calculation field contradicts that by providing a calculation.

d. Message: "IN THE STANDARDS FILE, LINE ____ SPECIFIES A STANDARD WHICH IS NOT BASIC BUT IT HAS NO CALUCLATION"

This means that for the indicated line in the standards file the standard number is not between 1 and 14, indicating that it is not a basic standard, but there was no calculation using basic standards included in the calculation field.

4.4 Special Maintenance Procedures. The normal procedure for executing a MACROSPITBOL program is to invoke the MACROSPITBOL interpreter with the command line: $ SPITBOL <filename>.SPT. To save execution time PFA uses the MACROSPITBOL feature of having the source code preinterpreted into a SNOBOL executable file which can then be loaded and run. (See EXIT(-1) in MACROSPITBOL Manual.) To facilitate the use of PFA, two command procedures are required in the user's LOGIN.COM file. The first one is a command to compile PFA programs: $ C*OMP :== $SYS$SYSTEM:SPITBOL/LOAD=PFAEDT NL:. This command (compile) can then be invoked by typing "C" or "COMP" by the user from command level. The command will then load and run a special PFA preprocessor called PFAEDT which processes (creates a MACROSPITBOL program) PFA source code and expands the "Included" files. PFAEDT then calls on the MACROSPITBOL interperter with the processed file, which then creates the saveable <filename>.SEX image file. The second command entry is $ PFA :== $SYS$SYSTEM: SPITBOL/LOAD=PFAINI NL:. This command, once in the LOGIN.COM file, is invoked by typing "PFA" at the command level. The command will then load and run a program called PFAINI, which presents a menu and asks for the PFA report program the user would like to run. PFAINI will then load and run the requested program which had been created by the compile command.

The programs PFAEDT.SPT and PFAINI.SPT used in the command procedures (C and PFA) must be compiled into loadable modules by the normal procedure outlined above. The rest of PFA can be compiled and run using only the C and PFA commands once they have been installed.

4.5 Special Maintenance Programs. None required.

4.6 Listings. Volume II, Appendix E, contains all program listings.

4.7 Software Failure Report Summary. The following is a summary of the existing known software failures and deficiencies:

4.7.1 Overall PFA. In list processing most lists use commas as item delimiters. However, lists of variables may include multi-subscripted array names which contain commas. It is suggested that percent signs be used in place of commas as primary list delimiters and that all subroutines used in list processing get as input the delimiter to be searched for.

4.7.2 STAMP.

a. READPATH uses start nodes to determine which subroutines are called by a task and its entry points, respectively. However, it assumes that a task's starting point will always be the first entry in the path table. This is not always the case: i.e., OUTOUT. It is suggested that the entry points in the structure have the same name as in the routine.

b. When outputting comments to HIPO charts, if the comment must be truncated, STROUT either truncates after the last comma or returns a null if there are no commas. Associated logic in OUTPATH may cause the assocated line of input variables to be lost. It is suggested that a separate routine be written to handle text.

4.7.3 PATCHANA.

a. If a module structure change is found, it is likely that everything after the structure change will be flagged because the structures of the two versions of the module will be out of synchronization. It is suggested that the PATCHANA program be changed to apply pattern matching to the structural patterns between parts of the two versions of the module being analyzed to determine the exact extent of the change in structure and to restore synchronization between the two versions so that additional structural changes can be identified. A percentage should also be calculated to indicate the amount or scope of the change.

b. PATCHANA does not recognize multiple entry points. This feature must be added.

4.8 Future Program Improvements. The following program improvements have been suggested but are not currently implemented.

4.8.1 Master File. In general, changes in the format of the master file will affect all PFA programs and encoders.

a. A new record type "D" should be defined for definition of global data, similar to labeled and blank commons in FORTRAN. This would provide better data tracking between modules.

b. Establish variable use codes for the variable records (type 7 records) as follows:

"D" - defined in an unexecutable statement

"M[C]" - modified, may be conditional

"R[C]" - referenced, may be conditional

"T[C]" - tested, may be conditional

"[L]" - loop control, suffix to above

"[U]" - undefined prior to use, suffix for above

c. A new record type "A" is needed to encode the algorithm used on the path segment in such a way as to allow theorem provers to be used where possible.

4.8.2 STAMP.

a. Names of modules of "blank common" and "labeled common" should be included in the module list and processed as modules.

TYPE OF MODULE	NAMING CONVENTION	TASK/SUBROUTINE CODE
Blank common	/*/ or / /	D
Named common	/name/	D
Executable routine	name	S, T, E

Commons may or may not be listed in the structure chart but will be listed in HIPO charts.

b. Version information should be deleted from the module list and be replaced with counts of subroutines calling each routine at each level of the subroutine hierarchy.

APPENDIX A

TERMS AND ABBREVIATIONS

TERMS AND ABBREVIATIONS

COMPRO	Comment Processor.
DEC	Digital Equipment Corporation.
DECUS	DEC User Society.
Encoder	A program which reads the source code for the system being analyzed and creates a master file for use by other PFA programs.
External procedures	Procedures which exist outside of the software being analyzed, e.g., operating system utilities.
HIPO	Hierarchy plus Inputs, Processing, and Outputs.
HIPO chart	A chart relating inputs and outputs to the processing algorithm which uses or creates them.
MACROSPITBOL	A SNOBOL4 compatible interpreter for the VAX.
Master file	The file which contains a representation of the attributes of the software being analyzed.
Module	A separately compilable procedure or subroutine.
Node	A point of decision in execution path selection.
PATCHANA	Patch Analysis Program.
Path	A segment of instructions which does not contain any branches.
PFA	Program Flow Analyzer.
PFALIB	PFA Library.
Program structure	Graph of the sequence of all possible paths which may be executed within a program.
Quantifiers	A value which quantifies or totals some attribute on a path.
RATFIV	A structured FORTRAN translator.

TERMS AND ABBREBIATIONS (Cont'd.)

SOFTPRO	Software Profile Program.
Software patch	A change in source code.
Software system	Consists of one or more computer programs which perform one or more related functions.
STAMP	Structure, Timing, Analysis, Modeling Program.
Standards file	File which contains user-entered standards in a prescribed format.
Structure chart	Shows a subroutine call hierarchy in the form of a tree graph.
SYSTRUCT	System Structure Comparison Program.
TECOM	Test and Evaluation Command.
USAEPG	U.S. Army Electronic Proving Ground.